WEB APLIKACIJA ZA POVEĆANJE UČINKOVITOSTI NABAVE REZERVNIH DIJELOVA ZA AUTOMOBILE NA TEMELJU ANALIZE KVAROVA

Kupanovac, Filip

SVEUČILIŠTE JOSIPA JURJA STOSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

WEB APLIKACIJA ZA POVEĆANJE UČINKOVITOSTI NABAVE REZERVNIH DIJELOVA ZA AUTOMOBILE NA TEMELJU ANALIZE KVAROVA

Završni rad

Filip Kupanovac

Osijek, 2021.

SADRŽAJ

1. UVOD 1
1.1. ZADATAK ZAVRŠNOG RADA 2
2. NABAVA REZERVNIH DIJELOVA NA TEMELJU ANALIZE KVAROVA I PRIKAZ STANJA U PODRUČJU 3
2.1. IZAZOVI NABAVE REZERVNIH DIJELOVA ZA AUTOMOBILE 4
2.2. PRIMJENA ANALIZE KVAROVA U UČINKOVITOSTI NABAVE REZERVNIH DIJELOVA 4
2.2.1. Analiza kvarova prema vrsti dijelova 4
2.2.2. Analiza kvarova prema proizvođačima 6
2.3. PRIKAZ POSTOJEĆIH POSTUPAKA I PROGRAMSKIH RJEŠENJA ZA NABAVU REZERVNIH DIJELOVA 8
2.3.1. Online Car Parts 9
2.3.2. EU Spares 11
3. MODEL I POSTUPCI NABAVE DIJELOVA ZA AUTOMOBILE 13
3.1 FUNKCIONALNI ZAHTJEVI WEB APLIKACIJE 13
3.2. NEFUNKCIONALNI ZAHTJEVI WEB APLIKACIJE 15
3.3. POSTUPAK ANALIZE KVAROVA I STVARANJE PREPORUKA NABAVE DIJELOVA 16
3.3.1. Protokol On-board diagnostic 16
3.3.2. Primjena dijagnostike protokolom OBD-II 17
4. PROGRAMSKO RJEŠENJE WEB APLIKACIJE 19
4.1. PROGRAMSKI JEZICI, TEHNOLOGIJE I RAZVOJNA OKOLINA 19
4.1.1. Opisni jezik HTML 19
4.1.2. Cascading Style Sheets 19
4.1.3. Programski jezik JavaScript 20
4.1.4. Node.js i Node package manager 20
4.1.5. Razvojna okolina Visual Studio Code 21
4.1.6. Baza podataka 22
4.2. PROGRAMSKO RJEŠENJE NA STRANI KORISNIKA 22
4.2.1. Postupak prijave korisnika 23
4.2.2. Korisničko sučelje vozača 25
4.2.3. Korisničko sučelje servisera 29
4.3. PROGRAMSKO RJEŠENJE NA STRANI POSLUŽITELJA 33
4.3.1. Baza podataka 34

4.3.2. Implementacija baze podataka putem poslužitelja 34
4.3.3. Programsko rješenje poslužitelja 35
5. KORIŠTENJE I ISPITIVANJE APLIKACIJE S ANALIZOM REZULTATA 38
5.1. NAČIN KORIŠTENJA WEB APLIKACIJE 38
5.1.1. Način rada vozača 38
5.1.2. Način rada servisera 41
5.2. ISPITIVANJE RADA APLIKACIJE 44
5.2.1. Korisnički slučaj 1 44
5.2.2. Korisnički slučaj 2 45
5.2.3. Korisnički slučaj 3 46
5.3. ANALIZA RADA APLIKACIJE 48
6. ZAKLJUČAK 49
LITERATURA 50
ŽIVOTOPIS 52
SAŽETAK 53
ABSTRACT 54
PRILOZI 55

1. UVOD

Prilikom prve kupnje automobila, većini kupaca je u primarnom interesu kupiti što bolje vozilo za što manje novaca. Osim samog vozila, potencijalnim kupcima preporučuje se da, osim cijene samog vozila, prouče i cijene rezervnih dijelova i troškova održavanja. U današnjem vremenu kada su automobili postali puno složeniji i napredniji, serviseri vozila u većini slučajeva nisu u mogućnosti samostalno i vlastoručno pregledati vozilo kako bi otkrili nastalu pogrešku i odredili koji je rezervni dio potrebno zamijeniti. Od velike važnosti i koristi u tim slučajevima dolazi dijagnostika vozila. Njena zadaća je analizirati vozilo i na temelju njegovog stanja otkriti i otkloniti nastali kvar. Nakon provedene dijagnostike, serviser će biti u mogućnosti vlasniku automobila predočiti stanje vozila i ukazati mu na trošak koji će biti uzrokovan nabavom potrebnih rezervnih dijelova.

Kako bi se postupak dijagnostike i uklanjanja kvara maksimalno ubrzao, potrebno je stvoriti aplikaciju koja će pružiti krajnjim potrošačima što veću razinu učinkovitosti nabave rezervnih dijelova upravo na temelju dijagnostike vozila. Povezivanjem dijagnostike automobila s uređajem za dijagnostiku putem protokola On-board diagnostic II, dijagnosticirati će se nastala pogreška te korištenjem aplikacije pronaći potreban postupak za otklanjanje nepravilnosti vozila. Ovim pristupom biti će omogućena nabava dijelova prije nastanka kvara čime se skraćuje vrijeme čekanja na rezervni dio po nastanku kvara te se tako povećava učinkovitost nabave dijelova, a vrijeme nedostupnosti vozila zbog kvara smanjuje.

U drugom poglavlju rada obraditi će se problematika nabave rezervnih dijelova na temelju analize kvarova te prikazati postojeći postupci i programska rješenja dostupna potrošačima za nabavu rezervnih dijelova. U trećem poglavlju obraditi će se model i postupci nabave dijelova za automobile kao i funkcionalni i nefunkcionalni zahtjevi web aplikacije, kao i postupak dijagnostike vozila i analize nastalih kvarova te način stvaranja preporuke za nabavu potrebnih rezervnih dijelova. Četvrtim poglavljem prikazat će se opis programskog rješenja web aplikacije teoretski opisom korištenih jezika, kao i programskim kodom na strani korisnika, ali i na strani poslužitelja. U posljednjem, petom poglavlju, opisat će se upute za korištenje same aplikacije te njezino ispitivanje i analiza rada nakon provedenih testova.

1.1. ZADATAK ZAVRŠNOG RADA

U teorijskom dijelu završnog rada treba proučiti i opisati probleme kvarova automobila, postupke analize i predviđanja kvarova, utjecaj kvarova na nabavu rezervnih dijelova, odnosno na lanac opskrbe, te dati pregled postojećih postupaka i programskih rješenja koji prema literaturi mogu povoljno utjecati na postupak nabave rezervnih dijelova na temelju predviđanja kvarova. Uz to, treba predložiti postupak analize kvarova i postupak stvaranja preporuka i obavijesti za nabavu dijelova. Također, treba predložiti model, dizajn i arhitekturu web aplikacije s bazom podataka koja će korisniku, odnosno vlasniku automobila s jedne i serviseru s druge strane omogućiti unos parametara, prikaz rezultata analize i predviđanja kvarova, te omogućiti učinkovito planiranje nabave dijelova za redovito održavanje i nastale kvarove u lancu opskrbe. Nakon kratkog opisa potrebnih programskih tehnologija, jezike i razvojne okoline, u praktičnom dijelu rada, treba programski ostvariti navedenu web aplikaciju na strani klijenta i poslužitelja, te je ispitati i analizirati za odgovarajuće ulazne podatke, scenarije kvarova i nabave rezervnih dijelova.

2. NABAVA REZERVNIH DIJELOVA NA TEMELJU ANALIZE KVAROVA I PRIKAZ STANJA U PODRUČJU

Porastom broja vozača posljednjih godina, i općenito korisnika automobila, raste ujedno i broj kvarova te nužnih servisiranja u određenim intervalima. Svaki od tih postupaka zahtjeva ujedno i izmjene određenih dijelova unutar samog vozila. To je potrebno obaviti zbog povećanja sigurnosti u prometu te mogućnosti da vlasnik automobila ima manje vremenskih perioda kada mu vozilo nije na raspolaganju zbog održavanja.

Kao svojevrstan pojam, kvar je definiran kao prestanak sposobnosti određenog elementa da izvršava funkciju koja se od njega zahtjeva i za koju je on namijenjen. Nakon nastupa samog kvara, element zadobiva pogrešku, koja može biti podijeljena na potpunu ili djelomičnu. Iz navedenog proizlazi zaključak kako je kvar samo događaj, za razliku od pogreške koja predstavlja svojevrsno stanje elementa na kojemu je nastupio kvar.

Kod ispravnih auto dijelova, pod utjecajem različitih čimbenika, nastupaju oštećenja. Samo oštećenje definirano je kao stanje u kojem određeni element (u ovom slučaju dio automobila) još uvijek ima sposobnost za izvođenje funkcije za koju je namijenjen, ali daljnjim korištenjem može nastupiti kvar - događaj u kojem on ne može više obavljati svoju zadanu funkciju. Nakon nastanka kvara, automobil prelazi iz stanja ispravnosti u stanje neispravnosti. Tada je potrebno na samom autu obaviti postupak popravka ili izmjene dijela pod kvarom. Izmjena takvih dijelova je češće korišten postupak od prepravka dijelova te ključni faktor postupka je tada dostupnost zamjenskog dijela [1].

Prilikom standardnih servisnih postupaka izmjene dijelova uglavnom je primijenjen ustaljen postupak radnji. Prema tom postupku prvo je dijagnosticiran kvar, zatim naručeni dijelovi od dobavljača ili direktno samog proizvođača i potom nakon nekoliko dana čekanja ostvarena je izmjena dijelova odnosno popravak automobila. Za vrijeme ovog postupka vlasniku automobil nije raspoloživ, čime on ostaje bez vlastitog prijevoznog sredstva na određeni broj dana koji u najvećoj mjeri ovise o brzini dostave traženog dijela. U današnjem užurbanom načinu života, s mnogo trenutno dostupnih usluga bez čekanja, ovakva iskustva vlasnicima su izrazito nepoželjna te se traže načini kako bi ona bila spriječena.

2.1. IZAZOVI NABAVE REZERVNIH DIJELOVA ZA AUTOMOBILE

Kako bi moglo biti ostvareno kraće vrijeme popravka vozila, potrebno je da mehaničari u svojem skladištu imaju na raspolaganju potrebne dijelove za slučaj kvara na automobilu. Obzirom na postojanje mnogo različitih proizvođača automobila, različitih dijelova izrađenih po drugačijim standardima i općenito velikog broja različitih modela automobila koji zahtijevaju personalizirane dijelove isključivo za taj model, skladištenje dijelova za različite modele bi zahtijevalo veliku količinu skladišnog prostora te financijskih sredstava za održavanje skladišta. Uz to, neki dijelovi vjerojatno ne bi bili korišteni često kao drugi te bi zbog stajanja mogli izgubiti određene kvalitete i pouzdanost, stoga ovakvo rješenje također nije prihvatljivo. Jedan od načina rješavanja ovog problema je korištenje strategije nabave Točno-na-Vrijeme (*eng. Just-in-Time - JIT*).

JIT strategija je prema [2] industrijski koncept pribavljanja raznih dijelova i sirovina kojim se postiže manja potreba za skladišnim kapacitetima na način da se potrebni materijali pribave neposredno prije njihove uporabe u poslovnom procesu. Ovim pristupom moguće je, korištenjem dijagnostike za analizu i predviđanje budućih popravaka, pravovremeno pribaviti potrebne dijelove u svrhu ostvarenja manjih čekanja na servis i postizanja više razine zadovoljstva korisnika uslugama.

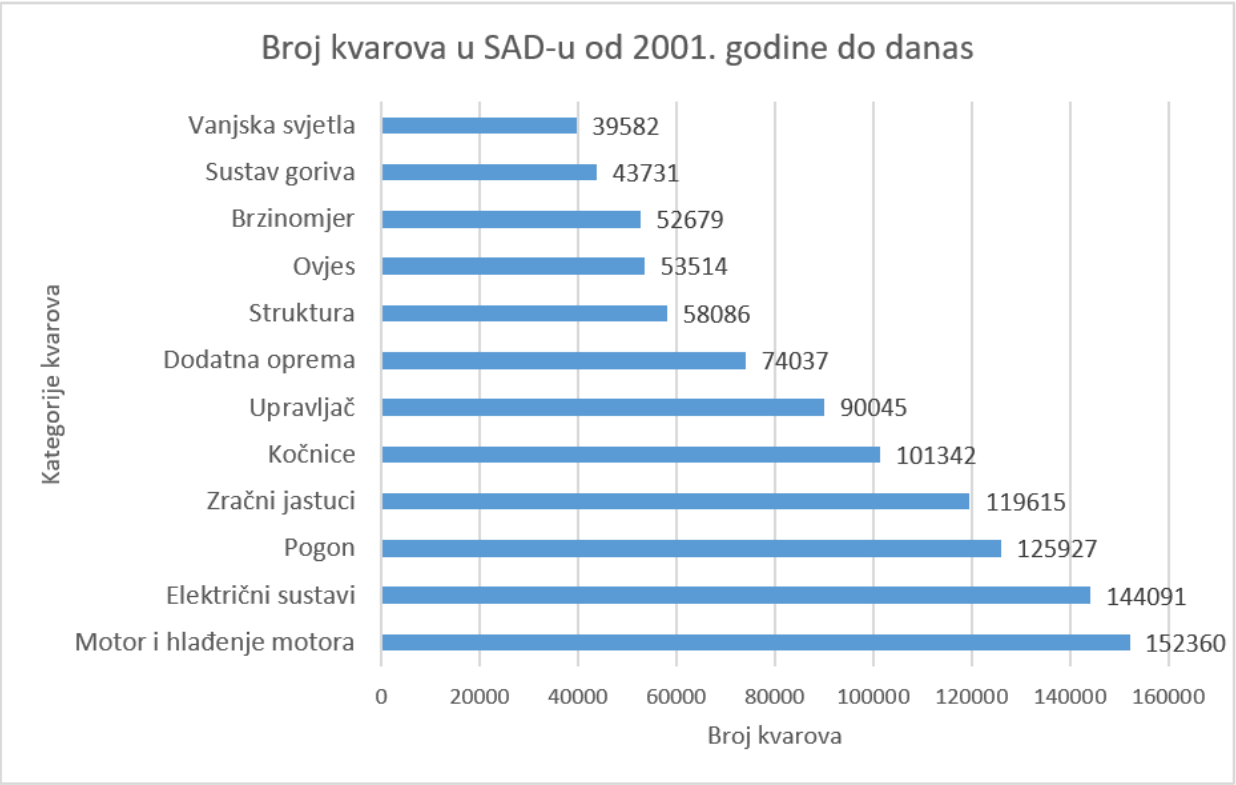
2.2. PRIMJENA ANALIZE KVAROVA U UČINKOVITOSTI NABAVE REZERVNIH DIJELOVA

Analiziranjem i statističkom obradom podataka o kvarovima na vozilima, moguće je utvrditi najčešće probleme te ih razvrstati ovisno o raznim faktorima poput korištenih dijelova, proizvođača, modela i slično. Grupiranjem kvarova na automobilima uočava se učestalost pojave istovrsnih kvarova među različitim grupama svrstanih po ranije spomenutim faktorima. Tako su uočeni češće prisutni kvarovi na automobilima istog proizvođača, modela ili čak serije proizvodnje u slučaju kada su pri izradi nove serije stvorena poboljšanja no zbog nekompatibilnosti s ostatkom sustava češće dovode automobil u stanje kvara. Primjenom analize učestalih kvarova moguće je predvidjeti buduće kvarove na automobilima koji se pojavljuju među grupom istovrsnih automobila.

2.2.1. Analiza kvarova prema vrsti dijelova

Prema istraživanjima web stranice CarProblemZoo.com [3], analizirani su najčešći kvarovi automobila na području SAD-a u zadnjih 25 godina. Koristeći podatke koji se svakodnevno

ažuriraju, web stranica prikuplja izvještaje vozača na području SAD-a o nastalim automobilskim kvarovima još od davne 1996. godine. Prema navedenoj analizi, pomoću prikupljenih podataka grafički su slikom 2.1 prikazani najčešći automobilski kvarovi u razdoblju od 2001. godine do danas (na dan 24. lipanj 2021.).



Slika 2.1. Najčešći automobilski kvarovi u razdoblju od 2001. godine do danas

Iz prikazanog grafikona na slici 2.1 može se uočiti kako najveći broj kvarova dolazi u području motora samog vozila te sustava za hlađenje. Osim toga, među zastupljenijim kvarovima su i kvar na električnom sustavu, pogon te kvar sa zračnim jastucima i kočnicama. S druge strane, najmanji broj kvarova zabilježen je na vanjskim svjetlima automobila te u sustavu goriva.

Osim navedenih kvarova, još neki od najčešćih su kvar na pojasevima, parkirnoj kočnici, unutarnja svjetla, dizelskom motoru te hibridnom pogonskom sustavu. Za mogućnost popravka ovih kvarova, od iznimne je važnosti industrija automobilskih dijelova koja, osim proizvođačima za direktnu ugradnju u vozila, svoje dijelove plasira na tržište kako bi bili dostupni za kupovinu kod nekog od mnogobrojnih prodavača. Industrija automobilskih dijelova i pribora jedan je od najizazovnijih dijelova automobilskog sektora. Nakon što je sama industrija pretrpjela posljedice globalne ekonomske recesije, ipak u narednim godinama nastupa njezin oporavak. Jedan od značajnih podataka je taj da je europska automobilska industrija rezervnih dijelova početkom

2021. godine dosegla vrijednost od 256 milijardi dolara, a s godišnjom stopom rasta od 3,1% očekuje se da će do 2026. godine dosegnuti vrijednost čak 349 milijardi američkih dolara.

2.2.2. Analiza kvarova prema proizvođačima

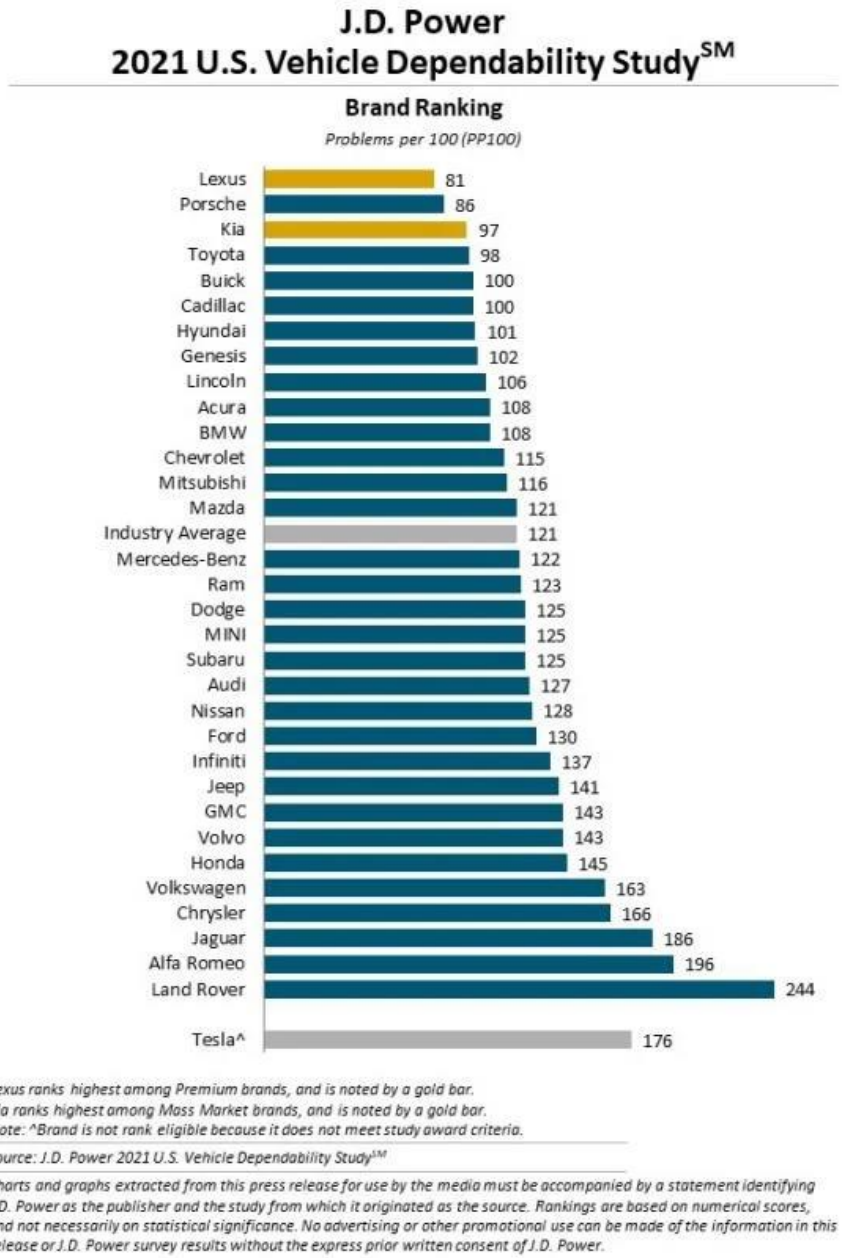
Kvarovi na automobilu se pojavljuju frekventnije kod nekih proizvođača u odnosu na druge. Razlog tomu su različite tehnologije i preciznost izrade automobila, kao i sama kvaliteta korištenih dijelova. Stoga je i tijekom samog procesa kupovine automobila kupcima važna njihova pouzdanost. Ta kvaliteta daje do znanja kako će automobil biti ispravan i zadovoljiti potrebe vlasnika. Problem se pojavljuje u tome što svaki pojedini vlasnik automobila pojam pouzdanog automobila tumači na vlastiti način. Primjer toga, vlasnici koji svoj automobil koriste uglavnom za vožnje u gradu mogu pouzdan automobil smatrati onaj koji ne zahtjeva ništa više od uobičajenih, odnosno redovnih popravaka (zamjena ulja, popravci kočnica, gume…). S druge strane vlasnicima koji automobil koriste i izvan gradova, u ruralnim i šumskim područjima treba snažniji automobil sa sposobnošću lakše i udobnije vožnje po nepravilnom terenu, te naravno, da automobil bude izdržljiv te može kroz duže vrijeme voziti takvim terenima bez kvara.

Unatoč tome što svaka od tih različitih grupa vozača zahtijeva različite karakteristike automobila, svima je zajednička želja i nastojanje za automobilom s minimalnim brojem kvarova kako bi on što češće bio na raspolaganju. S druge strane, nepouzdani automobil bi, prema tim tumačenjima, predstavljao automobil koji zahtjeva stalan broj neplaniranih kvarova.

Sama pouzdanost vozila može ostvariti ozbiljan utjecaj na zadovoljstvo potrošača, ali i vrijednost samog automobila pri njegovoj daljnjoj preprodaji. Upravo iz tog razloga potrošači nastoje kupovati automobile od proizvođača koji su, prema istraživanjima stručnjaka, ostvarili najbolju ocjenu u njihovoj pouzdanosti. Jedan od najvjernijih prikaza rezultata pouzdanosti automobila dolazi od strane tvrtke J. D. Power - američke tvrtke zadužene za analitiku podataka i potrošačku inteligenciju. Prema njihovoj američkoj studiji o pouzdanosti vozila objavljenoj u veljači 2021. godine [4], doneseno je nekoliko glavnih zaključaka:

- pouzdanost današnjih vozila dosegla je najveću razinu u povijesti automobilske industrije
- napredak kvalitete same proizvodnje predviđen je za kamionete i terence
- japanski i korejski proizvođači drže uvjerljivo dobru razinu proizvodnje
- poboljšavaju se vanjski izgled vozila i iskustvo u vožnji
- titulu najpouzdanijeg automobila ove godine osvojio je Porsche 911
- automobilski proizvođač Tesla prvi put je uvršten u analizu pouzdanosti

Rezultati istraživanja o pouzdanosti automobila izražavaju se pomoću pokazatelja PP100 (*Problems per 100*) koji predstavlja početnu kvalitetu određenu brojem problem na 100 vozila, pri čemu tada niži dobiveni rezultat predstavlja višu kvalitetu vozila. Na slici 2.2 preuzetoj iz [4], prikazani su rezultati nakon provedenog istraživanja:



Slika 2.2. Istraživanje pouzdanosti vozila putem broja problema na 100 vozila [4]

Iz prikazanih rezultata može se uočiti kako je proizvođač automobila Lexus ocijenjen kao najviši brend u ukupnoj pouzdanosti vozila, s ocjenom 81 PP100. Zanimljivo je za spomenuti kako je ovo deveti put u deset godina da Lexus zauzima najviše mjesto. Nakon njega, drugo mjesto zauzima Porsche, a slijede ga Kia, Toyota, Buick i Cadillac. U odnosu na prethodna istraživanja,

Kia pokazuje znatna poboljšanja, smanjenjem svojeg rezultata na testu za čak 35 bodova prema PP100 od 2020. godine. Isto tako, ovo je prvi put da se proizvođač Kia nalazi među najbolje ocijenjenim proizvođačima automobila globalnog tržišta. Od ostalih proizvođača koji također pokazuju znatno velik napredak u svojoj pouzdanosti su Cadillac, Acura, Hyundai i Mitsubishi.

Ovakva istraživanja na velikom broju vozilu daju statistički uvid u učestalost pojave kvara te se prema tome može očekivati češći kvar na lošije rangiranim vozilima prema istraživanju. Samim time, potrebno je biti spremniji za mogući nastanak kvara te češće poduzimati radnje prije nastanka kvara, što u prvom planu predstavlja nabavu i skladištenje većeg broja dijelova upravo za te automobile jer je izglednije da će oni biti prije korišteni nego dijelovi korišteni u pouzdanijim automobilima, uz pretpostavku jednakog broja vozila pouzdanih i nepouzdanih automobila.

2.3. PRIKAZ POSTOJEĆIH POSTUPAKA I PROGRAMSKIH RJEŠENJA ZA NABAVU REZERVNIH DIJELOVA

Industrija automobilskih dijelova i pribora jedan je od ključnih dijelova automobilske industrije. Proizvođačima automobila nudi gotova rješenja u obliku komponenti koje se ugrađuju izravno u nove automobile prilikom proizvodnje, ali i mnogobrojnim serviserima rezervne dijelove koji se umjesto dijelova u kvaru ugrađuju prilikom popravka.

Povezanost s drugom ranije navedenom skupinom izravnih korisnika, serviserima, ostvarena je uglavnom preko posrednika u obliku trgovina rezervnih dijelova, koje na jednom mjestu objedinjuju ponudu proizvoda raznih proizvođača automobilskih dijelova. Stoga, trgovine rezervnih dijelova za automobile predstavljaju vrlo važan segment u ovom sektoru.

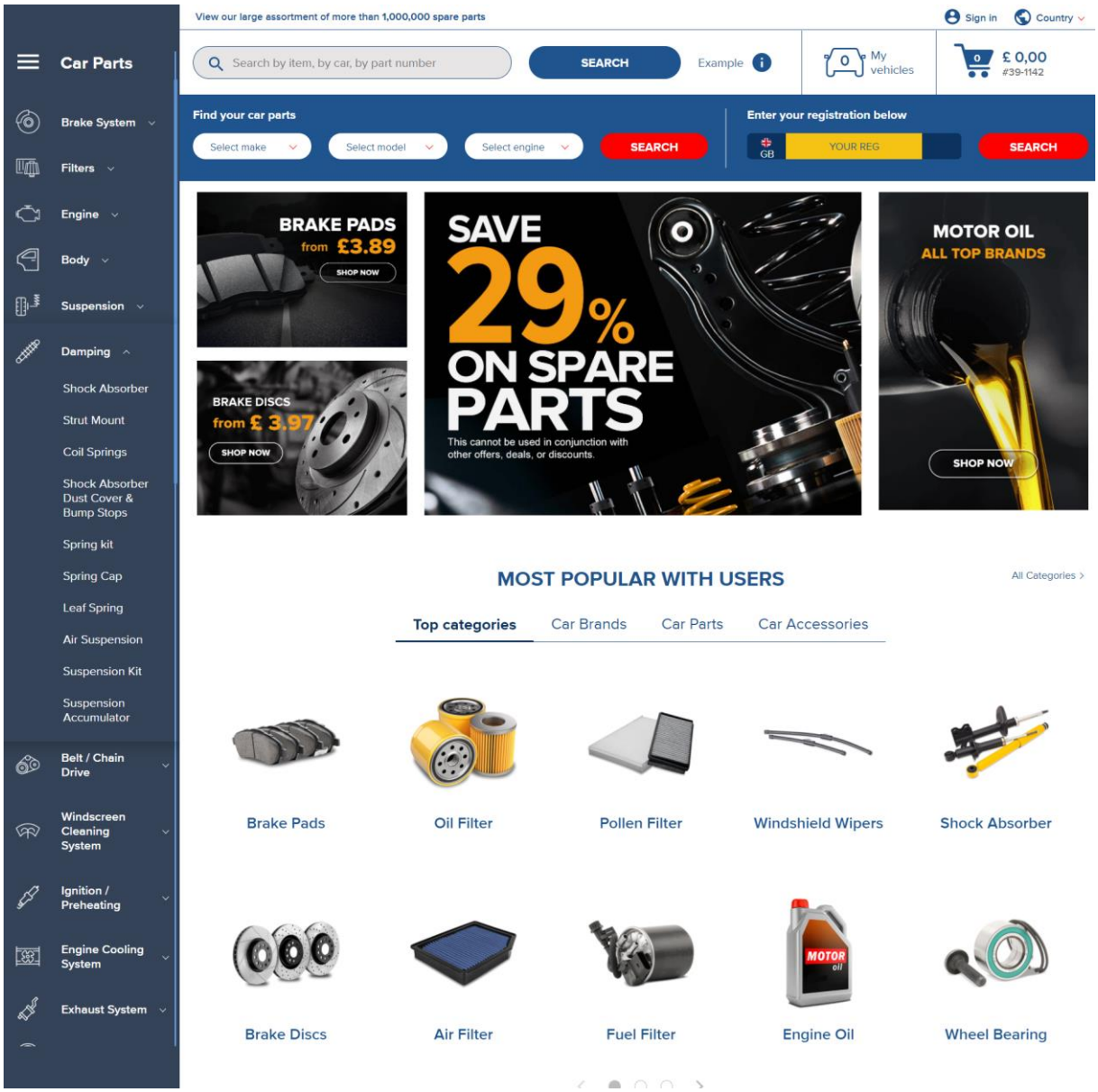
Unatoč brojnim izazovima nakon posljednje globalne recesije, posljednjeg desetljeća ovo tržište svake godine bilježi rast vrijednosti, kako na razini Europe, tako i na globalnoj razini. Također, nakon pada vrijednosti zbog globalne pandemije, brže od predviđenog dogodio se oporavak na tržištu te je početkom 2021. godine vrijednost europskog tržišta rezervnih dijelova dosegla razinu iz 2019. godine s procjenom vrijednosti od 256 milijardi američkih dolara (USD). Rastući trend vrijednosti se previđa i u bliskoj budućnosti te je predviđena kumulativna godišnja stopa rasta od 3.1% prema kojoj bi tržište dosegnulo vrijednost od 369 milijardi USD [5]. Digitalizacijom tržišta postavljeni su novi trendovi među potrošačima te je prisutan porast udjela e-tržišta u ukupnom tržištu rezervnih dijelova. Zbog ponude šireg opsega proizvoda različitih kvaliteta i mogućnošću zadovoljavanja potreba većeg broja potrošača internetske trgovine mogu

očekivati najveći dio očekivanog porasta vrijednosti cijelog tržišta, do rasta vrijednosti čak 4 puta prema procjenama [6].

2.3.1. Online Car Parts

Online Car Parts [7] je internetska trgovina dijelova za automobile sa središtem registriranim u Berlinu, no sva prodaja dijelova se vrši isključivo online, dok je ciljano tržište kompanije ono europsko. U ponudi se nalazi preko milijun različitih proizvoda od više od 500 različitih proizvođača, čime uspješno pokrivaju zahtjeve velikog udjela potrošača. Skladišni kapaciteti smješteni su u samom središtu Europe, u Njemačkoj čime je ostvarena mogućnost relativno brze dostave u bilo koji dio ciljanog tržišta.

Na početnoj stranici trgovine, prikazanoj na slici 2.3 nude se različiti kriteriji za pretragu dijelova. Moguće je pretragu započeti prema osnovnim sustavima unutar automobila te zatim detaljnijom klasifikacijom izabrati određeni dio i nakon toga potražiti prema modelu automobila dio koji pristaje. Druga opcija je pretraživanje prema modelu auta gdje se unosi proizvođač, model i tip motora, nakon čega korisniku bivaju ponuđeni isključivo dijelovi kompatibilni s odabranim automobilom. Sljedeća opcija je unos registracije gdje se prema registraciji pronalazi automobil te su ponovno ponuđeni dijelovi isključivo za taj automobil. Još jedna opcija za pretragu je izravno upisom gdje se prema unesenim riječima pretražuje baza i nude dijelovi koji odgovaraju unesenim parametrima po bilo kojem atributu (naziv, proizvođač, model). Posljednja mogućnost dostupna korisnicima je izrada korisničkog računa kojim nakon prijave mogu unijeti svoje automobile u vozni park čime je ponuđena brža alternativa traženju dijelova isključivo za odabrani automobil.



Slika 2.3. Početnica stranica Online Car Parts trgovine

Osim ponude dijelova, trgovina prikazuje i opće informacije o nekom dijelu prilikom odabira konkretnog dijela iz izbornika.

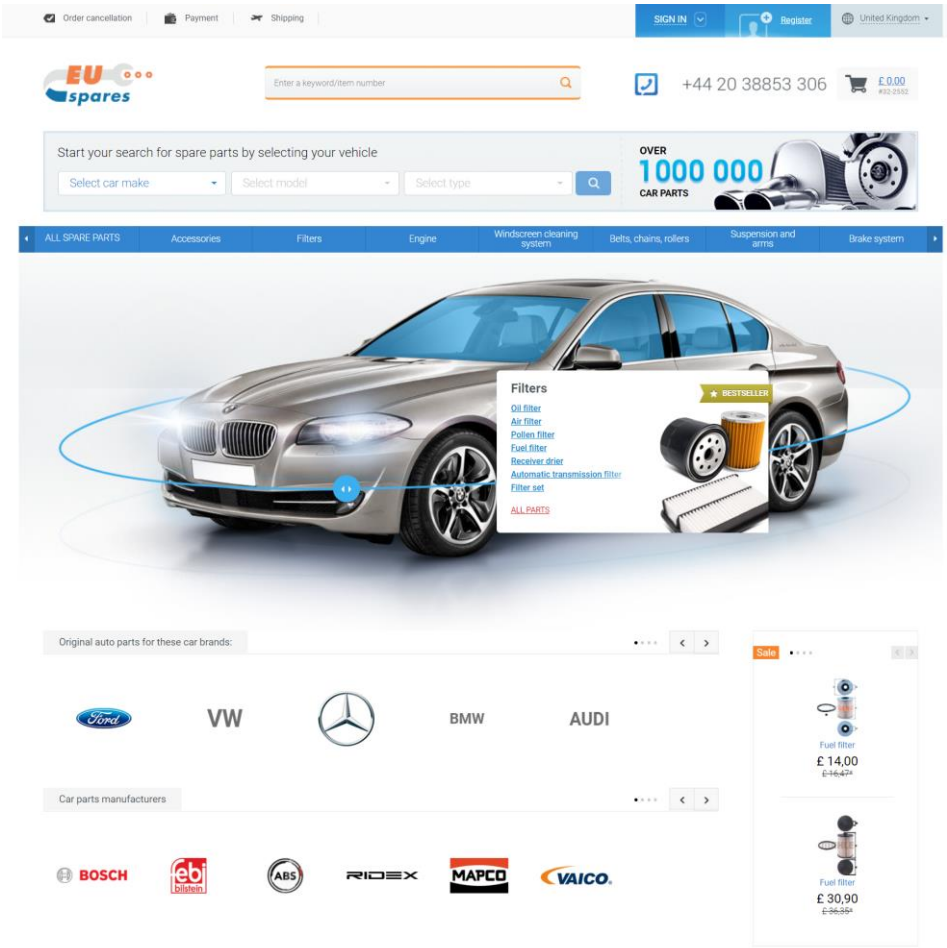
Ponuđene informacije uključuju:

- opis dijela
- pomoć za prepoznavanje oštećenja ili kvara
- moguće razloge kvara
- popis nužnih koraka pri popravku
- opis postupka zamjene dijela novim.

2.3.2. EU Spares

EU Spares [8] je trgovina sa središtem u Berlinu u vlasništvu grupacije Wemax. Koristeći geografski povoljnu lokaciju skladišta također lociranih u Njemačkoj, omogućavaju brzu isporuku dijelova na ciljano tržište. Svoje proizvode, uključujući Europu, dostavljaju u više od 80 država svijeta, uz opciju besplatne dostave za robu vrijednosti iznad 140 britanskih funti.

Pretraga dijelova je omogućena na više različitih načina. Prvi od njih je unos ključnih riječi ili broj artikla. Zatim je moguće tražiti i prema vrsti proizvoda počevši od sustava kojem traženi dio pripada te nadalje detaljnijim odabirom kategorija i potkategorija potražiti određeni dio. Moguće je i unijeti model automobila te tražiti dijelove isključivo kompatibilne s odabranim modelom. Još jedna opcija pretraživanja započinje odabirom proizvođača vozila te prikazuje izbor dijelova kompatibilnih s odabranim proizvođačem vozila. Uz to, moguće je tražiti prema proizvođaču dijelova te se tada prikazuje asortiman dijelova isključivo odabranog proizvođača, što je vidljivo na slici 2.4.



Slika 2.4. EU Spares početna stranica

Registrirani korisnici prijavom na svoj profil imaju mogućnost pregleda prošlih narudžbi i stvaranja liste želja gdje mogu spremiti dijelove te im kasnije brže pristupiti u slučaju narudžbe. Većina različitih vrsta dijelova sadrži i informacije o tom dijelu koje se prikažu odabirom određenog dijela. Informacije o dijelu uključuju opis samog dijela i njegove uloge u automobilu, popis vodećih proizvođača tog dijela te očekivani životni vijek dijela pri korištenju u normalnim uvjetima.

3. MODEL I POSTUPCI NABAVE DIJELOVA ZA AUTOMOBILE

Unutar ovog poglavlja teorijski će biti opisani postupci i funkcionalnosti koji će kasnije biti provedeni u djelo aplikacijom i nefunkcionalni zahtjevi za provjeru kvalitete izvedbe aplikacije. Opisati će se i ranije spomenuti protokol dijagnostike, On-board diagnostic II.

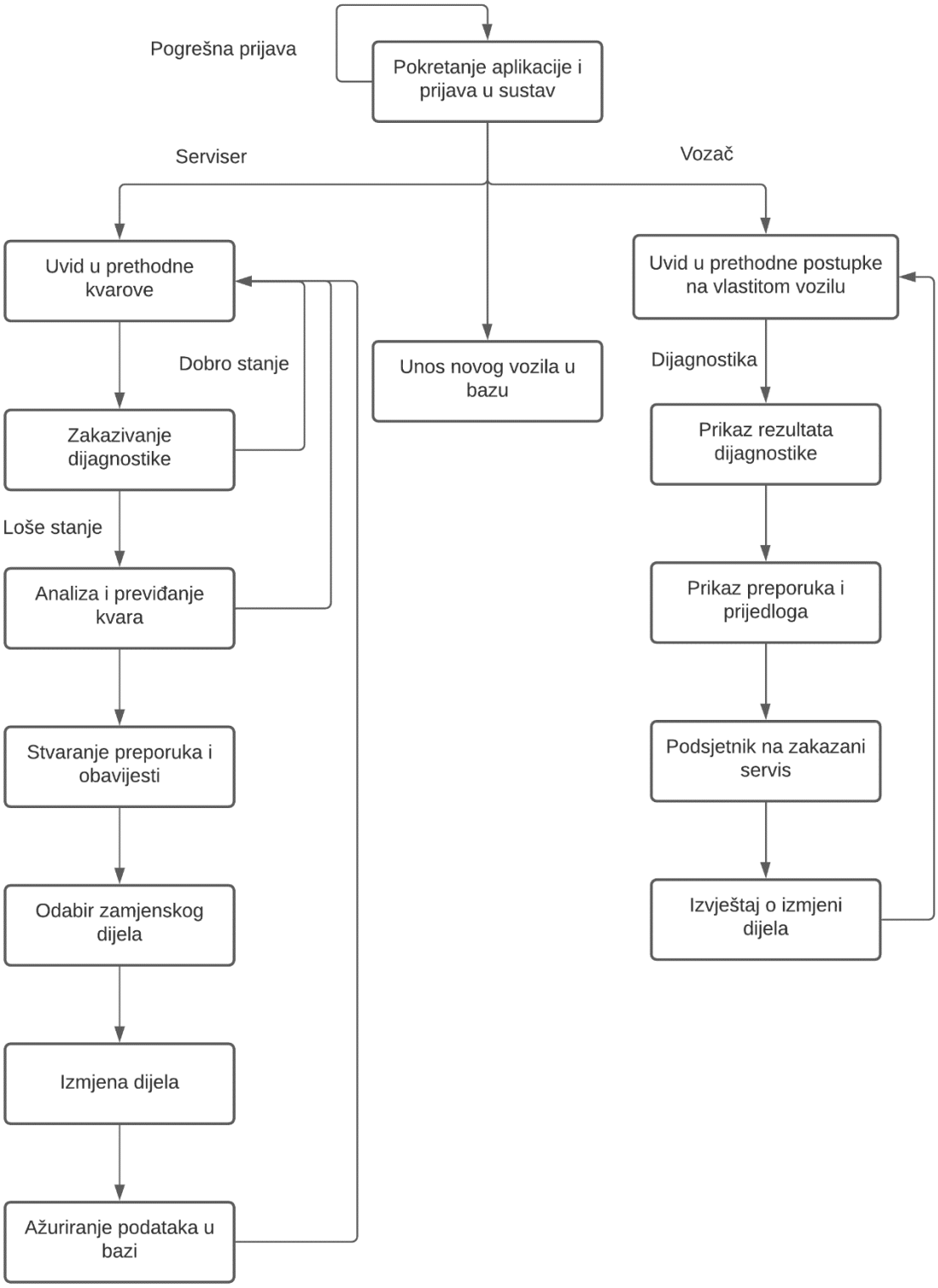
3.1 FUNKCIONALNI ZAHTJEVI WEB APLIKACIJE

Funkcionalni zahtjevi neke aplikacije predstavljaju zahtjeve pri izradi aplikacije koje ona mora ispuniti. Ti zahtjevi se vežu uz funkcionalnosti aplikacije te opisuju glavne radnje koje se poduzimaju prilikom korištenja aplikacije i oni su temelj rada iste.

Najvažnije funkcionalnosti ove aplikacije su:

- Stvaranje korisničkog računa i prijava korisnika
- Ovisno o vrsti korisnika (vozač/serviser), omogućavanje funkcionalnosti prema profilu korisnika
- Unos vozila u bazu podataka zajedno s pripadnim primarnim serviserom
- Prikaz prethodnih kvarova i servisnih postupaka na vozilima te mogućnost uvida u kvarove i servisne postupke istovrsnih modela vozila
- Mogućnost zakazivanja dijagnostike prema problemu koje vozilo ima
- Analiza i predviđanje kvarova na temelju statističkih pokazatelja
- Stvaranje preporuka i obavijesti za nabavu dijelova na temelju analize i predviđanja kvarova
- Odabir zamjenskog dijela u skladu s potrebnom razinom kvalitete i prihvatljivom cijenom
- Po završetku postupka, serviser unosi podatke o dijagnostici, servisu i nabavi dijelova u bazu
- U slučaju otkrivanja serijske pogreške u proizvodnji, svim vlasnicima tog modela pristiže upozorenje o izmjeni te je nužno odmah dogovoriti termin zamjene neispravnog dijela
- Vozaču omogućiti uvid u stanje za vlastiti automobil

Na slici 3.1 je prikazan dijagram toka svih funkcionalnosti koje aplikacija izvršava prema navedenim zahtjevima.



Slika 3.1. Dijagram toka aplikacije

Aplikacija nakon uspješne prijave pruža korisniku sučelje ovisno o vrsti profila, a dvije moguće vrste su serviser i vozač. Profil servisera ima veće ovlasti unutar aplikacije iz razloga što ima veću odgovornost pred sobom. On može vidjeti prethodne kvarove i postupke na svim

automobilima kojima je radio servis te imati uvid u najčešće kvarove i pouzdanost tih modela vozila, što ostaje skriveno vozaču. Prilikom nastanka pogreške na nekom automobilu, serviser određuje termin dijagnostike gdje se pruža bolji uvid u stvarno stanje automobila u realnom vremenu. Po završenom postupku dijagnostike, analiziraju se rezultati te ukoliko je loše stanje aplikacija predviđa sljedeći mogući kvar. Zatim i serviser i vozač bivaju obaviješteni o rezultatima te se prikazuje preporuka za zamjenu dijela s vjerojatnošću nastanka kvara. Tada je potrebno odabrati zamjenski dio koji se naručuje kako bi serviser bio spreman na izvršavanje servisa i zamjenu dijela u što kraćem vremenskom periodu kada se ukaže potreba, ili preventivno prema dogovoru. Na kraju cjelokupnog postupka serviser treba unijeti podatke o servisu u bazu.

Kao što je vidljivo sa slike 3.1, vozač ima mnogo manje ovlasti te su mu ponuđene informacije samo o vlastitom automobilu na uvid. Vozač je unutar aplikacije ovlašten za dodavanje novog vlastitog automobila u aplikaciju, te potraživanje termina dijagnostike kada uoči nepravilnost u radu vlastitog vozila.

3.2. NEFUNKCIONALNI ZAHTJEVI WEB APLIKACIJE

Nefunkcionalni zahtjevi aplikacije predstavljaju zahtjeve koji opisuju koliko dobro, odnosno učinkovito sustav izvršava zadane funkcionalne zahtjeve (izvršava funkcionalnosti). Tipični nefunkcionalni zahtjevi postavljeni aplikacijama su: pouzdanost, preglednost podataka, izvedivost, održivost, upotrebljivost, dokazivost kroz testove, sigurnost, proširivost, performanse aplikacije i mnogi drugi [9]. Prema ovim zahtjevima i njihovom ispunjavanju ocjenjuje se kvaliteta aplikacije te je pri izradi ideje aplikacije uz funkcionalne, vrlo bitno dobro definirati i željene nefunkcionalne zahtjeve.

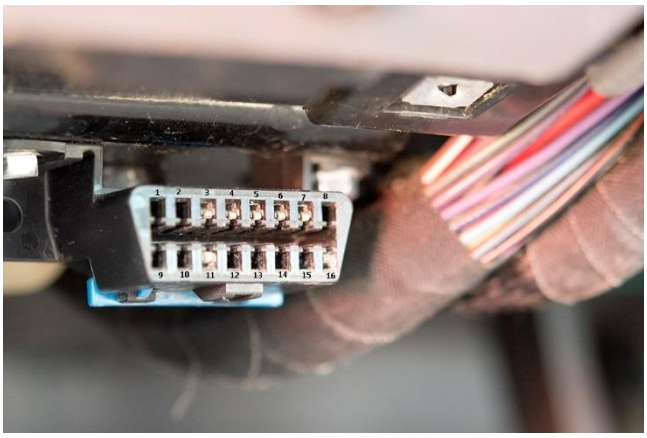
Ovoj aplikaciji postavljeni su zahtjevi proširivosti, performansi i održivosti, uz dokazivost kroz testove. Iako trenutno izrađena za malen broj testnih korisnika, moguće je implementirati aplikaciju za veći broj korisnika, kao i proširiti bazu rezervnih dijelova i dijagnostičkih kodova te tako proširiti mogućnosti korištenja aplikacije. Kvaliteta performansi označava da se stranica može brzo učitati na većini uređaja, prema [7] unutar 3 sekunde u normalnim uvjetima, odnosno 5 sekundi pri opterećenju sustava. Jednostavnim dizajnom ostvaruje se ovaj zahtjev unutar aplikacije. Korištenjem načela objektno-orijentiranog programiranja ostvaruje se bolja održivost sustava čime je otvorena mogućnost jednostavnog dodavanja novih funkcionalnosti ukoliko tijekom životnog vijeka nastanu novi funkcionalni zahtjevi. Testiranjem i prikazom rezultata testiranja aplikacije unutar poglavlja 5, dokazano je i kako je aplikacija dokaziva kroz testiranje.

3.3. POSTUPAK ANALIZE KVAROVA I STVARANJE PREPORUKA NABAVE DIJELOVA

Automobili od svojeg nastanka i početka serijske proizvodnje početkom prošlog stoljeća konstantno dobivaju nova poboljšanja i značajke. Proizvođači ih na razne načine pokušavaju unaprijediti te se taj proces može usporediti s evolucijom – traže se bolje i nove mogućnosti, inovacije i primjene raznih ideja kako bi automobili postali sve privlačniji, ugodniji i jednostavniji za korištenje, dok oni koji ne uspijevaju uhvatiti korak s vremenom i ponuditi nova rješenja trpe gubitke prilikom kretanja tržišta. Tako su vremenom automobili prerasli u vozila s mnoštvom elektronike, pokretana središnjim računalom koje putem tisuća raznih senzora nadgleda i upravlja radom automobila. Pošto se korištenjem sve većeg broja dijelova i uređaja u automobilu povećava i mogućnost pogreške ili kvara nekog od njih, a time i „zaključavanja“ automobila da on ne može raditi dok se problem ne otkloni, stvorena je potreba za što bržim pregledom i analizom stanja automobila.

3.3.1. Protokol On-board diagnostic

Serviser danas u većini slučajeva ne može sam ručno pregledati automobil kako bi otkrio nastalu pogrešku, pogotovo ukoliko je ona nastala na elektroničkoj komponenti. Stoga su razvijena razna rješenja za analizu stanja automobila koja pomažu u otkrivanju i učinkovitijem otklanjanju pogreške. Najčešće korišteno rješenje je protokol On-board diagnostics druge generacije (OBD-II). Ovaj protokol koristi komunikaciju računala u automobilu s krajnjim izvršnim dijelovima i senzorima te analizira rezultate njihove komunikacije. Tako se u vrlo kratkom roku mogu provjeriti mnogi sustavi unutar automobila, a prema zaprimljenim vrijednostima, uređaj za analizu putem protokola OBD-II vraća korisniku izvještaj o dijelu s pogreškom [10]. Počevši od 1996. godine, sva vozila proizvedena u Sjedinjenim Američkim Državama moraju biti kompatibilna s ovim protokolom, dok isto vrijedi za vozila izrađena u Europskoj Uniji od 2004. godine. Uređaj za dijagnostiku spaja se na računalo automobila putem posebnog priključka sa 16 pinova. Stoga i automobili moraju imati ugrađen konektor kompatibilan za korištenje protokola OBD-II prikazanog na slici 3.2 koji je definiran standardom SAE J1962.



Slika 3.2. automobilski priključak za OBD-II

3.3.2. Primjena dijagnostike protokolom OBD-II

Spajanjem dijagnostičkog uređaja za analizu putem OBD-II, korisniku se nudi više različitih načina rada standardiziranih standardom SAE J1979 / ISO 15031-5 [11]. Oni uključuju korištenje raznih alata za analizu rezultata, prikupljanje uzoraka brojčanih vrijednosti, praćenje rada automobila u stacionarnom stanju ili čak prilikom vožnje. Ukoliko se ne želi prikupljati vremenski uzorak za detaljnu analizu performansi, učinkovitosti ili drugih pokazatelja, moguće je koristiti način rada izravnog prikaza dijagnostičkog koda kvara (*eng. Diagnostic Trouble Code – DTC*) kojim se po pokretanju rada šalju signali iz uređaja za dijagnostiku prema računalu automobila. Računalo zatim šalje kontinuirane impulse prema izvršnim uređajima i senzorima automobila kojima provjerava ispravnost dijelova. Po povratku impulsa natrag do računala, uređaj za dijagnostiku prikuplja rezultate te prema njima na svojem zaslonu prikazuje rezultat dijagnostike u obliku koda koji se sastoji od jednog slova te četiri brojčane znamenke [12].

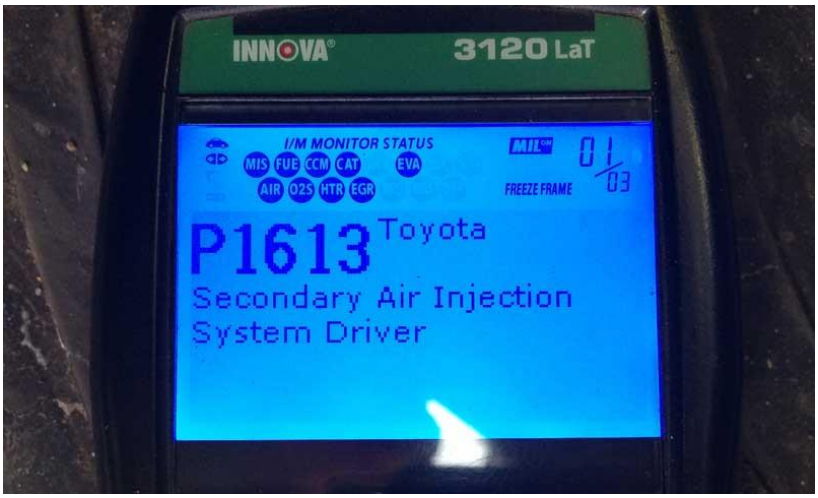
Početno slovo koda pogreške označava sustav unutar automobila koji je neispravan. Može poprimiti 4 različitih vrijednosti, a svako slovo se odnosi na zasebni sustav:

- Pxxxx – pogon (*eng. powertrain*)
- Bxxxx – karoserija (*eng. body*)
- Cxxxx – šasija (*eng. chassis*)
- Uxxxx – korisnička mreža (*eng. user network*)

Sljedeća znamenka poprima jednu od dvije moguće vrijednosti – 0 ili 1, koje predstavljaju generičke kodove važeće za sve automobile (0) te specijalne kodove karakteristične za pojedinog proizvođača (1).

Treća znamenka koda pogreške predstavlja podsustav unutar sustava definiranog prvom znamenkom te je za svaki osnovni sustav različito tumačenje podsustava. Posljednje dvije

znamenke označavaju samu pogrešku unutar prikazanog sustava i podsustava [13]. Pomoću potpunog koda dobivenog na zaslonu uređaja moguće je otkriti specifičnu pogrešku na automobilu u svrhu otklanjanja greške ili zamjene dijela kako bi automobil ostao ispravan i siguran za daljnje korištenje. Primjer očitanja pogreške prikazan je na slici 3.3 u nastavku.



Slika 3.3. Primjer dijagnostičkog koda pogreške na zaslonu uređaja

Unosom koda pogreške očitanog s uređaja za dijagnostiku unutar aplikacije, povezuje se očitani kod s problemom iz baze podataka te daje na uvid mehaničaru. To omogućava ispravan odabir dijela za zamjenu, te se može izvršiti narudžba kako bi servis mogao biti što prije učinjen.

4. PROGRAMSKO RJEŠENJE WEB APLIKACIJE

U ovome poglavlju rada teoretski su opisani korišteni jezici u izradi aplikacije su potom prikazane stvorene funkcionalnosti u okviru aplikacije uz popratna objašnjenja te ulogu koju imaju u radu aplikacije.

4.1. PROGRAMSKI JEZICI, TEHNOLOGIJE I RAZVOJNA OKOLINA

4.1.1. Opisni jezik HTML

HTML (*HyperText Markup Language*) predstavlja opisni jezik za prikaz sadržaja na Internetu [14]. Služi za definiranje strukture i sadržaja internetskih stranica i aplikacija te je osnova na kojoj se dalje grade i unaprjeđuju internetske stranice. Nakon nekoliko godina rada na hipertekstualnim (*eng. hypertext*) sustavima namijenjenim dijeljenju datoteka među zaposlenicima unutar CERN organizacije [15], Tim Berners-Lee je osmislio prvu verziju jezika prilagođenu za široku upotrebu na internetu i objavio ju 1993. godine.

Glavno obilježje jezika su elementi koji određuju strukturu web stranice. Elementi se sastoje od oznake i sadržaja. Oznake elementa se navode između kutnih zagrada te uobičajeno imaju početnu i završnu oznaku između kojih se nalazi sadržaj. Ipak, mogući su i elementi bez završne oznake. Prilikom prikaza HTML dokumenta na Web pregledniku, korisniku se ne prikazuju oznake, već samo sadržaj koji se nalazi između njih.

4.1.2. Cascading Style Sheets

Cascading Style Sheets (CSS) je stilski jezik namijenjen vizualnom uređivanju web stranica i sadržaja na istima. Isprva je osmišljen kako bi se Web stranice mogle stilski urediti zasebno izvan HTML dokumenta, kako je to prije nastanka CSS-a bila praksa [16]. Na ovaj način razdvojeni su struktura sadržaja stranice od stilskog prikaza i prezentacije same stranice unutar zasebnih dokumenata tako da svaka vrsta dokumenta ima isključivo jednu ulogu. Prvu verziju CSS-a objavio je World Wide Web Consortium (W3C), tijelo koje se bavi razvojem i prihvaćanjem Web standarda, krajem 1996. godine [17].

Stilove je moguće putem CSS koda primijeniti na tri različita načina:

- stil unutar linije (*eng. inline style*)
- unutarnja lista stilova (*eng. internal stylesheet*)
- vanjska lista stilova (*eng. external stylesheet*)

Sintaksa CSS-a podrazumijeva navođenje selektora – elementa na koji se deklaracija stila odnosi, a može biti oznaka elementa, klasa ili identifikator, zajedno uz kombinacije više njih – te nakon odabranog selektora deklaraciju stilova unutar vitičastih zagrada u parovima svojstvo-vrijednost (svojstvo : vrijednost;). Ukoliko se za isti element definira više različitih vrijednosti istog svojstva, kasnije navedena vrijednost će biti prihvaćena jer pri implementaciji stranice primjenjuju stilove po redu kako su napisane u listi, a u slučaju konflikta stara vrijednost se samo prepiše novom vrijednosti.

4.1.3. Programski jezik JavaScript

Programski jezik JavaScript nastao je 1995. godine, tijekom rane faze razvoja interneta, s namjerom „oživljavanja“ web stranica, odnosno dodavanjem programa na, do tada, statične web stranice bez funkcionalnosti. Njegovom pojavom omogućena je interakcija na web stranicama bez potrebe ponovnog učitavanja cijele stranice za svaku radnju. Prvotni naziv jezika trebao je biti LiveScript, no zbog tadašnje popularnosti programskog jezika Java, razvojni tim Netscape Navigator preglednika se odlučio na ime JavaScript jer su unutar preglednika podržavali Javu, ali i zbog boljeg plasmana jezika na tržište. Podržava koncept objektno-orijentiranog programiranja, kao i funkcionalnog stila. Osim na samim preglednicima, JavaScript kod se može izvršiti na bilo kojem uređaju koji u sebi sadrži JavaScript pokretač, a danas je prisutan u više od 95% Web stranica [18].

Iznimno brzim porastom popularnosti, JavaScript je već 1996. godine imao veliki broj preglednika koji su ga podržavali i bilo je nužno postaviti standard u razvoju kako bi svi preglednici jednako interpretirali jezik. Odgovornost za održavanje predana je ECMA-i (*eng. European Computer Manufacturers Association*) koja i danas razvija jezik i proširuje njegove mogućnosti. Tada je ujedno i službeno promijenjen naziv u ECMAScript, no i dalje se popularno naziva prema izvornom imenu. Daljnjim razvojem nastajale su proširene verzije jezika u razmacima od nekoliko godina, no pravu revoluciju izazvala je verzija ECMAScript 6 2015. godine s mnoštvom novih mogućnosti, i promjenom u načinu održavanja pa otada svake godine nastaju manja ažuriranja umjesto dotadašnjih većih promjena svakih nekoliko godina.

4.1.4. Node.js i Node package manager

Node.js je JavaScript pokretač napravljen za omogućavanje stvaranja skalabilnih aplikacija korištenjem programskog jezika JavaScript na Web preglednicima, ali i izvan njih. Omogućuje

korisnicima pisanje programskog koda kako za korisničku stranu (*eng. front-end*), tako i za poslužiteljsku stranu (*eng. back-end*). Mogućnosti prilikom korištenja ovakvog pristupa su mnogobrojne, a ovise samo o uključenim modulima, odnosno programskim paketima i bibliotekama [19].

Mnogi programski paketi i biblioteke mogu se naći unutar Node package managera (npm). Npm se sastoji od registra s javno dostupnim (*eng. open-source*) paketima, kao i onih čije je korištenje potrebno platiti. Sadrži i vlastiti naredbeni redak (*eng. command line client*) za pristup željenim paketima. Za potrebe ovog rada korišteni su paketi React, Express i Knex.

React je JavaScript biblioteka namijenjena stvaranju interaktivnog korisničkog sučelja aplikacije koju održavaju Facebook i zajednica mnogobrojnih programera i tvrtki. Sučelje izrađeno pomoću Reacta se dijeli na manje komponente s ciljem da se funkcionalnosti razdvoje na što manje jedinice koda, zvane komponente. Komponente se dijele na funkcijske i klasne. Funkcijske se deklariraju poput funkcija te samo vraćaju JSX (JavaScript XML) elemente koji čine sučelje, dok su klasne stvorene poput klasa s vlastitim stanjem i ponašanjem, nasljeđuju roditeljsku klasu *Component* te obavezno moraju prepisati metodu *render* koja vraća JSX. Osim te metode, moguće je definirati vlastite korisničke metode raznih funkcionalnosti. JSX koristi elemente definirane HTML-om uz mogućnost korisničkog definiranja novih elemenata, koje korisnik može stvoriti primjenom HTML elemenata.

Express je biblioteka otvorenog koda namijenjena za korištenje na poslužiteljskoj strani. Služi za rukovanje HTTP (*Hipertext Transfer Protocol*) zahtjevima. Radi svoje fleksibilnosti korisnicima pruža širok spektar mogućnosti prilikom stvaranja vlastitog poslužitelja ili sučelja za programiranje aplikacija (*eng. Aplication programming interface – API*).

Knex je biblioteka namijenjena korištenju na poslužiteljskoj strani za stvaranje SQL (*eng. Structured Query Language*) upita korištenih u bazama podataka. Sintaksa biblioteke je slična SQL jeziku, uz određene prilagodbe prema JavaScript standardu. Dijelovi SQL naredbi se putem Knex-a slažu poput asinkronog koda s objektima tipa *Promise* gdje svaka ključna riječ iz jezika SQL predstavlja *callback* tip metode s odabranim parametrom kao argumentom metode.

4.1.5. Razvojna okolina Visual Studio Code

Visual Studio Code je besplatni uređivač koda razvijen od strane Microsofta za korištenje na Windows, macOS i Linux operacijskim sustavima. Objavljen je 2015. godine s većinom

izvornog koda objavljenog na GitHub repozitoriju. Značajke koje nudi ovaj uređivač su podrška za otklanjanje grešaka (*eng. debugging*), označavanje različitih kategorija riječi koda drugim bojama (*eng. syntax highlighting* - ključne riječi, varijable, funkcije,…), inteligentno dovršavanje koda, refaktoriranje koda i ugrađena podrška za Git, odnosno kontrolu verzija koda. Osim za Git, ima ugrađenu podršku za jezike JavaScript, TypeScript i Node.js, ali nudi i mnoštvo ekstenzija za druge popularne programske jezike.

Veliku popularnost ostvaruje jer je lagan i moćan, ali i zbog ekstenzija pomoću kojih korisnici mogu prema želji urediti izgled, podržane značajke i personalizirati program prema željama i potrebama uz zadržavanje osnovnih značajki. Ekstenzije u trgovini VS Code-a može objaviti svaki autor putem svojeg računa i tako ih dati na korištenje ostalim korisnicima. Po instalaciji, ekstenzije je moguće omogućiti, onemogućiti, ažurirati i ukloniti, kako bi se prilagodio program ovisno o potrebama projekta.

4.1.6. Baza podataka

Obzirom na nedostupnost otvorenog i javno dostupnog sučelja za programiranje aplikacija ili baze podataka, kako za dijagnostičke kodove protokola OBD-II tako i za zamjenske dijelove, u svrhu izrade web aplikacije stvorene su tablice unutar baze podataka koje sadrže potrebne informacije o dijagnostičkim kodovima (tablica *diagnostic_codes*) i zamjenskim dijelovima (tablica *spare_parts*). Svi podaci unutar tablica su uneseni prema podacima s Web stranica [6] i [11]. Radi opsežnosti količine sadržaja, unutar tablica su prisutni samo neki izdvojeni podaci potrebni za prikaz rada aplikacije.

4.2. PROGRAMSKO RJEŠENJE NA STRANI KORISNIKA

Ovo potpoglavlje rada sadrži detaljan prikaz ostvarenja funkcionalnosti aplikacije navedenih u potpoglavlju 3.1 za sve zahtjeve postavljene na aplikaciju, uz popratni programski kôd korišten za njihovo ostvarenje. Pokretanjem aplikacije stvara se instanca klase *App*. To je početna točka aplikacije na kojoj se stvara korisničko sučelje prema daljnjem tijeku aktivnosti u aplikaciji, čiji je prikaz sadržaja određen i sadržan u metodi *render()* prikazanoj programskim kodom 4.1.

```
class App extends Component {
  constructor(){
    super();
    this.state = {
      route: 'signin',
      isSignedIn : false,
      user: {
        email: '',
        name: '',
        type: '',
        id: ''
  }}}
  render(){
    return(
      <>
        <Navigation isSignedIn={this.state.isSignedIn} signOut={this.SignOut} setRoute={this.SetRoute}/>
        <div className='content'>
          <>{this.GetContent()}</>
        </div>
      </>
    )
  }
  GetContent(){
    switch(this.state.route){
      case 'signin': return <Signin loadUser={this.LoadUser} trySignIn={this.TrySignin}/>
      case 'register': return <Register loadUser={this.LoadUser} trySignIn={this.TrySignin}/>
      case 'driv': return <Driver user={this.state.user}/>
      case 'mech': return <Mechanic user={this.state.user}/>
      default: return <></>
    }
  }
```

Programski kod 4.1. Klasa *App*, početna točka aplikacije

4.2.1. Postupak prijave korisnika

Na početnom sadržaju aplikacije nalazi se obrazac za prijavu u sustav (*eng. Sign in*) prema kojemu je nazvana komponenta *Signin*. Sučelje te komponente se sastoji od dva obavezna polja za unos podataka, jednog tipa *email* za unos računa elektroničke pošte te *password* za unos lozinke korisnika. Uz ta dva polja, nalazi se i gumb za pokušaj prijave u sustav. Po pritisku tog gumba, aplikacija provjerava jesu li unesene informacije u oba polja za unos podataka te ukoliko nijedno nije ostavljeno nepopunjeno, šalje zahtjev tipa *POST* poslužitelju u čijem se tijelu nalazi sadržaj iz polja za unos (programski kod 4.2.). Ukoliko poslužitelj u bazi podataka pronađe korisnika sa zaprimljenim podacima za prijavu, aplikacija zaprima natrag podatke o korisniku iz baze te se ovisno o vrsti korisnika (vozač ili serviser) prikazuje korisničko sučelje te vrste korisnika.

```
Submit = () => {
    fetch('http://localhost:3000/signin', {
        method: 'post',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({
            email: this.state.email,
            password: this.state.password
        })
    }).then(res => {
        let odg = res.json();
        return odg;
    }).then(user => {
        if(user !== 'wrong credentials'){
            this.props.loadUser(user);
            this.props.trySignIn()
        }
    }).catch(err => {
        console.log(err)
    })
}
```

Programski kod 4.2. Metoda za prijavu postojećeg korisnika u sustav

Korisnicima koji se po prvi puta prijavljuju u sustav, ponuđena je mogućnost registracije kako bi sami stvorili korisnički račun. Uz desni rub navigacijske trake na vrhu stranice, nalazi se tekst *Register* koji na klik preusmjerava korisnika na sučelje za registraciju novih korisnika, čija je izvedba prikazana kodom 4.3.

Vizualno slično sučelje onome za prijavu postojećih korisnika u sustav, uz već navedena polja za unos elektroničke pošte i lozinke, sadrži i polje za unos korisničkog imena koje će se prikazivati u aplikaciji za tog korisnika. Klikom na gumb za registraciju ispod obrasca, šalje se zahtjev za stvaranjem novog računa te ukoliko ne postoji račun koji je prijavljen na navedenu adresu elektroničke pošte, stvara se novi korisnički račun.

Osim tri polja za unos, postrance se nalazi i potvrdni okvir s tekstom koji upućuje na registraciju računa mehaničara. Potvrdom te stavke, na ekranu se stvara novo polje za unos validacijskog koda mehaničara. Ovim postupkom aplikacija nudi stvaranje novog računa mehaničara, no unos pogrešnog validacijskog koda uz označen potvrdni okvir uzrokovati će pogrešan zahtjev za stvaranje računa čime se onemogućava slučajno stvaranje računa tipa vozač prilikom pokušaja stvaranja računa mehaničara.

```
render(){
    return(
        <div>
            <fieldset>
                <p>Registracija</p>
                <div className="register-form">
                    <div className="half-width">
                        <div className="fill-width pa05 flex">
                            <label>Korisničko ime</label>
                            <input onChange={this.onNameChange}
                                type="text" name="name"
                            ></input>
                        </div>
                        <div className="fill-width pa05 flex">
                            <label>Email</label>
                            <input onChange={this.onEmailChange}
                                type="email" name="email-address"
                            ></input>
                        </div>
                        <div className="fill-width pa05 flex">
                            <label>Lozinka</label>
                            <input onChange={this.onPasswordChange}
                                type="password" name="password"
                            ></input>
                        </div>
                    </div>
                    <div className="half-width">
                        <div className="fill-width pa05">
                            <input onChange={this.onCheckboxCheck}
                                type="checkbox" name="mechanic-check"
                                value="mechanic"
                            ></input>
                            <label name="mechanic-check">Ja sam mehaničar</label>
                        </div>
                        <>{this.ShowMechanicInputField()}</>
                    </div>
                </div>
            </fieldset>
            <button onClick={this.Register}>
                Registriraj se
            </button>
        </div>
    )
}
```

Programski kod 4.3. Definiranje korisničkog sučelja za registraciju korisnika

4.2.2. Korisničko sučelje vozača

Po uspješnoj prijavi korisničkog računa za profil vozača, glavna komponenta aplikacije, *App* preusmjerava prikaz karakterističan za vozače, korištenjem komponente *Driver*. U toj komponenti je definirano sučelje vozača, koje nudi pregled i funkcionalnosti koje su mu aplikacijom omogućene.

Na vrhu sučelja vozača ponuđen je gumb za dodavanje novog vozila kojim korisnik sam može unijeti svoje vozilo u aplikaciju te se uneseno vozilo pridodaje njegovom računu u bazi

aplikacije. Za spremanje vozila u bazu potrebno je unijeti njegov serijski broj, proizvođača, model, godinu proizvodnje, vrstu pogona, snagu izraženu u konjskim snagama te registracijsku oznaku.

Sva vozila prijavljenog vozača prikazana su jedno do drugoga u obliku kartica koje sadržavaju informacije o samom vozilu. O ovoj funkcionalnosti se brine komponenta *VehicleCard*, prikazana programskim kodom 4.4, koja se koristi unutar komponente *Driver*. Po uspješnoj prijavi vozača i prikazu sadržaja za korisnika, poslužitelju se šalje zahtjev metodom tipa *GET* za dohvaćanje svih vozila čiji je vlasnik prijavljeni korisnik te se prema podacima dohvaćenim iz odgovora poslužitelja stvaraju navedene kartice.

```
class VehicleList extends Component {
    constructor(props){
        super(props);
        this.state = {
            ownerId: this.props.id,
            ownerCars: [],
            count: 0
        }
    }

    render(){
        return(
            <div className="vehicle-list">
                <>{this.IterateCars()}</>
                <>{this.OnVehicleCountChange()}</>
            </div>
        )
    }
    componentDidMount(){
        this.FetchCars();
    }
    FetchCars = () => {
        fetch(`http://localhost:3000/cars/${this.state.ownerId}`)
        .then(response => response.json())
        .then(cars => {
            this.setState({ownerCars: cars})
            this.setState({count:cars.length})
        })
    }
    IterateCars = () => {
        if(Array.isArray(this.state.ownerCars)){
            return(
                <>  {this.state.ownerCars.map(car =>{
                        return(
                            <Vehicle key={car.serial_number} car={car}
                                pickvehicle={this.props.pickvehicle}
                            />
                        )
                    })}
                </>
            )
        }
        else
            return <></>
    }
```

Programski kod 4.4. Klasa za dohvaćanje vozila prijavljenog vozača

Iz prikazanog koda 4.4 vidljivo je kako svako vozilo čija je kartica stvorena unutar metode *IterateCars()* sadrži i argument *pickvehicle*. Taj argument je referenca na metodu *PickVehicle()* iz

roditeljske klase *Driver*, koja se proteže kroz argumente konstruktora komponenti sve do komponente *Vehicle*. Klikom na bilo koju instancu te komponente, aktivira se metoda kojom se odabire vozilo te se po odabiru u stanje roditeljske klase spremaju podaci o odabranom vozilu. Ponovnim klikom na vozilo koje je bilo odabrano prije klika, poništava se odabir vozila.

Podatak o odabranom vozilu je važan roditeljskoj klasi iz razloga što se podaci o odabranom vozilu šalju drugom djetetu klase *Driver*, komponenti *SubmitScheduleForm*. Navedena komponenta služi kao obrazac putem kojega vozač ima mogućnost zatražiti dijagnostički pregled kada za to osjeti potrebu, za vozilo koje je prethodno odabrao. Obrazac nudi 3 polja za unos podataka koja vozaču nude odabir željenog datuma dijagnostike, mehaničara te neobavezno polje za opis problema kako bi mehaničar već unaprijed mogao dobiti neke informacije od vozača. Prilikom unosa željenog servisera, dinamički se mijenja padajući izbornik koji nakon svake promjene unesene vrijednosti, prilikom koje aplikacija prikazuje sve servisere čiji se naziv podudara s istom kako bi korisnik lakše mogao odabrati servisera kod kojega želi obaviti pregled.

Klikom na gumb „Pošalji zahtjev“, poziva se metoda *TrySubmitSchedule()* prikazana kodom 4.5 koja provjerava ispunjenost svih potrebnih podataka za zahtjev te uz ispravno popunjene sve potrebne podatke, šalje zahtjev putem metode tipa *POST* poslužitelju za unos u bazu. Zahtjev je potom prikazan korisniku unutar okvira s navedenim svim zatraženim zahtjevima koji čekaju odobrenje servisera. Isti zahtjev se tada prikazuje odabranom serviseru, što je detaljnije opisano u odjeljku 4.2.3.

```
TrySubmitSchedule = () => {
    let {pickedVehicle,mechanicInput, dateInput} = this.state;
    if(pickedVehicle !== undefined && mechanicInput !== undefined
        && dateInput !== undefined && dateInput !== "")
    {
        this.SendScheduleRequest();
        this.ToggleSubmitAvailable();
    }
}
SendScheduleRequest = () =>{
    fetch(`http://localhost:3000/request-schedule`, {
        method: 'post',
        headers: {"Content-Type": "application/json"},
        body: JSON.stringify({
            mechanic: this.state.mechanicInput.id,
            vehicle: this.state.pickedVehicle.serial_number,
            date: this.state.dateInput,
            note: this.state.problemInput
        })
    })
    .then(resp => resp.json())
    .then(data =>{
        if(data.status !== 400){
            this.setState({dateInput: undefined,
                mechanicInput:undefined,
                problemInput: undefined})
            this.setState({key: Math.random()})
        }
    })
}
```

Programski kod 4.5. Metoda za stvaranje novog zahtjeva za pregled

Svaki nepotvrđeni zahtjev vozača za pregledom prikazan je unutar okvira s naslovom „Zatraženi pregledi“ koji prikazuje vozilo za koje je tražen pregled, traženi datum pregleda, te opis problema ukoliko je navedem (programski kod 4.6). Po potvrdi zahtjeva mehaničara, zahtjev se prebacuje u odjeljak „Termini dijagnostike“ gdje vozač može vidjeti potvrđeni termin dijagnostike, jednakog sučelja i prikaza informacija o terminu.

```
const AppointmentCard = ({appointment}) =>{
    let date = new Date(appointment.scheduled_time)
    return(
        <div className="appointment-driver">
            <hr/>
            <p>{appointment.manufacturer} {appointment.model}</p>
            <p>Serviser: {appointment.name}</p>
            <p>Termin: {date.getDate()}.{date.getMonth() +1}.{date.getFullYear()}.</p>
            <>{appointment.note !== null
             ? <p>Opis problema: {appointment.note}</p>
             : <></>
            }</>
        </div>
    )
}
```

Programski kod 4.6. Prikaz informacija pojedinog zahtjeva

Dok je postupak u tijeku, vlasniku je prikazano više informacija, kao što je dijagnostički kod pogreške, opis iste te predloženi postupak za rješavanje zajedno s prikazom vrste potrebnog zamjenskog dijela. Po narudžbi rezervnog dijela vozaču se mijenja prikaz te umjesto prikaza potrebnog dijela, prikazuju se informacije o specifičnom naručenom dijelu kojim će dio s pogreškom u vozilu biti zamijenjen.

Na posljednjem dijelu sučelja, vozač ima mogućnost vidjeti sve već dovršene postupke na vlastitim vozilima gdje su ponuđene glavne informacije o servisu, poput opisa pogreške, servisera koji je obavio postupak i zamijenjenog dijela. Programski kod pojedinog postupka dan je programskim kodom 4.7.

```
render(){
    let {appointment} = this.props
    return(
        <div className="appointment-driver">
            <hr/>
            <p>{appointment.manufacturer} {appointment.model}</p>
            <div className="register-form">
                <p className="pa-r nomarg">Dijagnostički kod pogreške: {appointment.code}</p>
                <p className="nomarg">Opis pogreške: {appointment.description}</p>
            </div>
            <p>Serviser: {appointment.name}</p>
            <>{appointment.code === "P0000" ? <p>Pregled. Nije pronađena pogreška na automobilu.</p>
                                            : <>{this.ShowService()}</>
            }
            </>
            <>{this.ShowPart()}</>
        </div>
    )
}
ShowService = () =>{
    let {appointment} = this.props;
    return(<p>Učinjeni postupak: {appointment.service_note}</p>)
}
ShowPart = () =>{
    let {part} = this.state;
    let {appointment} = this.props;
    return(
        <>{(appointment.made_order !== 'N' && part !==undefined)
        ?   <>  <p>Zamijenjeni dio: {part.service_part} {part.part_manufacturer}, EAN: {part.ean}</p>
                <p>Cijena izmijenjenog dijela: {part.price} GBP</p>
            </>
        :   <></>
        }</>
)}
```

Programski kod 4.7. Prikaz informacija o dovršenom servisu

4.2.3. Korisničko sučelje servisera

Serviser predstavlja u aplikaciji bitno drugačiji tip profila od vozača. Njegova uloga je odlučiti konačni termin dijagnostike i servisa, po odrađenom dijagnostičkom postupku unijeti dijagnostičke informacije te po urađenom servisu unijeti podatke o izmjeni dijela automobila.

Jednako kao i vozaču, serviseru je omogućeno dodavanje vozila u bazu podataka. Razlika u odnosu na vozačevu mogućnost je što serviser ne unosi svoje vozilo, već vozilo drugog korisnika. Stoga uz podatke koje sadrži i obrazac za unos vozila na strani vozača, serviser za dodavanje treba unijeti identifikacijsku oznaku vozača čije vozilo unosi u sustav. Programski kod za unos vozila od strane servisera dan je u nastavku.

```
AddNewVehicle = () =>{
    let {driverId,manufacturer,model,year,serial,drivetrain,horsepower,license}=this.state;
    if(this.ManufacModelCheck() && this.YearCheck()
        && this.SerialCheck() && this.DrivetrainCheck()
        && horsepower>0 && license !== undefined
        && driverId !== undefined)
    {
        fetch(`http://localhost:3000/add-vehicle`, {
            method:'post',
            headers:{'Content-Type':'application/json'},
            body: JSON.stringify({
                id: driverId,
                manufacturer: manufacturer,
                model: model,
                year: year,
                serial: serial,
                drivetrain: drivetrain,
                horsepower: horsepower,
                license: license
            })
        })
        .then(res => res.json())
        .then(data => {
            if(data.status !== 400)
                this.props.setFlag()
        })
        .catch(err => console.log)
    }
}
```

Programski kod 4.8. Slanje zahtjeva servisera za unos novog vozila

U sklopu korisničkog sučelja servisera, nalazi se popis svih zatraženih termina za pregled koje su razni vozači uputili serviseru. Odjeljak sa zatraženim terminima za svaki zahtjev prikazuje informacije o vozaču koji je zatražio pregled, vozilu za koje je pregled zatražen, željenim datumom i naposljetku opisom problema, ukoliko je naveden. Klikom na neki od zahtjeva, on postaje aktivan te se serviseru otvara mogućnost potvrde ili odbijanja zahtjeva klikom na neki od gumbova koji se prikažu na dnu zahtjeva. Također, ukoliko želi prihvatiti zahtjev, no traženi termin ne odgovara ili nije dostupan iz bilo kojeg razloga, serviser može potvrditi drugi datum, unutar vremenskog perioda od trenutnog dana sve do dva mjeseca kasnije. Po potvrdi zahtjeva, on prelazi u stanje čekanja dijagnostike te je na sučelju servisera prikazan u odjeljku dijagnostičkih zahvata na

čekanju. Primjerom programskog koda 4.9 prikazano je dohvaćanje svih postupaka na čekanju, spremanje istih u stanje klase, te metoda *ShowAwaitingDiagnostics()* kojom se isti prikazuju na sučelju. Spomenuti odjeljak djeluje slično odjeljku zatraženih zahvata, prikazuje jednake informacije te ima mogućnost odabira aktivnog.

```
FetchAwaitingDiagnostics = () =>{
    fetch(`http://localhost:3000/awaiting-diagnostics/${this.state.id}`)
    .then(response => response.json())
    .then(data => {
        this.setState({count: data.length})
        this.setState({awaitingDiagnostics: data})
    })
}
ShowAwaitingDiagnostics = () => {
    if(this.state.count){
        if(this.state.showAwaitDiagnostics){
            return(
                <><p className="pointer"
                    onClick={() => {
                        this.props.setFlag()
                        this.ToggleShowAwaitDiags()}}>
                    Prikaži manje</p>
                    {this.state.awaitingDiagnostics.map(diag => {
                        return(
                            <DiagnosticCard key={diag.appointment_number}
                                diagnostic={diag}
                                pickDiagnostic={this.props.pickDiagnostic}
                                pickedDiagnostic={this.props.pickedDiagnostic}
                                runDiagnostic={this.props.runDiagnostic}
                                />
                            )})}</>
        )}
        else{
            return(<p className="pointer"
                onClick={() => {
                    this.props.setFlag()
                    this.ToggleShowAwaitDiags()}}>
                Prikaži više</p>)
        }
    }
    else return <></>
}
```

Programski kod 4.9. Dohvaćanje i prikaz dijagnostika na čekanju

Po odabiru nekog dijagnostičkog zahvata, prikaz se proširuje tako da ispod informacija o zahvatu stoji polje za unos koda pogreške očitanog s uređaja za dijagnosticiranje kvara na automobilu. Po kliku na gumb *Potvrdi*, pokreće se metoda *RunDiagnostic(code)* čiji argument predstavlja dijagnostički kod pogreške (prikazano kodom 4.10). Ukoliko je uneseni dijagnostički kod valjan, odnosno postoji u bazi zajedno s vlastitim opisom problema pokreće se metoda *StartDiagnostic(code)* kojom se pridružuje rezultat dijagnostike odabranom zahvatu. Tada zahvat prelazi u stanje aktivnog postupka, gdje se prikazuje serviseru opis problema zajedno s potrebnim postupkom za otklanjanje u novom dijelu sučelja, odjeljku aktivnih postupaka.

```
PickDiagnostic = (pick) => {
    if(this.state.pickedDiagnostic !== undefined){
        if(pick.appointment_number !== this.state.pickedDiagnostic.appointment_number){
            this.setState({pickedDiagnostic: pick})}
        else{
            this.setState({pickedDiagnostic: undefined})}
    }
    else{this.setState({pickedDiagnostic: pick})}
}
RunDiagnostic = (code) => {
    fetch(`http://localhost:3000/diagnostic-code/${code}`)
    .then(res => res.json())
    .then(data =>{
        if(data !== "Code not found"){
            this.StartDiagnostics(data.code);
        }
    })
}
StartDiagnostics = (code) => {
    let {pickedDiagnostic} = this.state;
    fetch('http://localhost:3000/resolve-diagnostic', {
        method: 'put',
        headers: {'Content-Type':'application/json'},
        body: JSON.stringify({
            appointment_number: pickedDiagnostic.appointment_number,
            code: code
        })
    })
    .then(res => res.json())
    .then(data =>{
        this.SetFlag();
    })
}
```

Programski kod 4.10. Slanje dijagnostičkog koda i dohvaćanje informacija o pogrešci

Po uspješno izvršenoj dijagnostici, utvrđuje se nastala pogreška i potrebno je odabrati zamjenski dio za otklanjanje pogreške. Prema marki automobila pretražuje se baza dijelova one vrste koja je potrebna za servis te odgovara marki automobila na servisu, što je prikazano u programskom kodu 4.11. U dogovoru s vozačem se naručuje zamjenski dio te po narudžbi umjesto popisa dostupnih dijelova, prikazane su informacije o naručenom dijelu. Onoga trenutka kada je zamjenski dio dostavljen, može se izvršiti servis. Po završenom servisu, serviser treba unijeti izvještaj o provedenom postupku te zaključiti servis nakon kojega bi automobil trebao opet biti u ispravnom stanju.

```
GetParts = () =>{
    let {diagnostic, car} = this.props;
    fetch(`http://localhost:3000/parts/${diagnostic.service_part}/${car.manufacturer}`)
    .then(res => res.json())
    .then(data => this.setState({spareParts: data}))
}
ShowParts = () => {
    let {diagnostic} = this.props;
    let {spareParts, pickedPart} = this.state
    return(
        <>
            <div className="just-center">
                <p>Potrebni su vam dijelovi tipa: {diagnostic.service_part}</p>
            </div>
            <div className="just-center">
                <div className="flex space-around w70 wrap">{spareParts.map(part =>{
                    return(<SparePart key={part.ean}
                                sparePart={part}
                                pickPart={this.PickPart}/>)
                    })}
                </div>
            </div>
            <>{pickedPart !== undefined
            ?   <div className="just-center">
                    <div className="w70 in-progress">
                        <p>Odabrali ste dio: {pickedPart.service_part} {pickedPart.
                        part_manufacturer}, EAN: {pickedPart.ean}</p>
                        <p>Cijena: {pickedPart.price}GBP</p>
                        <div className="just-center">
                            <button
                                onClick={() => {
                                    this.OrderPart()}}
                            >Naruči</button>
                        </div>
                    </div>
                </div>
            : <></>
            }
            </>
        </>
    )
}
```

Programski kod 4.11. Dohvaćanje i prikaz rezervnih dijelova

Svi završeni servisi se prikazuju serviseru za evidenciju unutar kartice završeni servisi, definirane klasom *ResolvedAppointments* te su unutar iste za svaki servisni postupak prikazane informacije o svakom izvršenom servisu.

4.3. PROGRAMSKO RJEŠENJE NA STRANI POSLUŽITELJA

Poslužitelj u računarstvu predstavlja udaljeno računalo (ili skup više računala) koje pruža usluge krajnjim korisnicima. Korisnici šalju zahtjeve za podacima poslužitelja te po primitku zahtjeva, poslužitelj na njih odgovara pružanjem podataka na uvid ili izmjenu, ovisno o definiranim mogućnostima komunikacije poslužitelja i korisnika. Navedeni podaci uobičajeno su spremljeni unutar baze podataka te ih poslužitelj dohvaća i prosljeđuje korisnicima.

4.3.1. Baza podataka

Porastom veličine programa, broja korisnika i različitih sadržaja kojima poslužitelji barataju, nastala je potreba za učinkovitim sustavom prikupljanja i spremanja podataka, kao i razdvajanjem aplikacije od skupa podataka kojima se koristi. Sustav koji rješava ovaj problem je poznat kao sustav za upravljanje bazama podataka (*eng. DBMS – Database Management System*). Baza podataka predstavlja alat za organiziranje podataka i njihovo razdvajanje u smislene manje cjeline, a podaci koje mogu sadržavati su raznoliki te se struktura svake baze mijenja ovisno o potrebama aplikacije u kojoj se koriste [20]. Razvojem znanosti koja se bavi podacima, nastala su četiri glavna tipa baza podataka:

- hijerarhijski
- mrežni
- relacijski
- objektni

Ova aplikacija koristi relacijsku bazu podataka PostgreSQL. Obilježja takvih baza su relacijske tablice sa stupcima istovrsnih podataka, tablice su međusobno povezane stranim ključevima drugih tablica te sadrže vlastiti primarni ključ, dok je jezik za pisanje instrukcija unutar baze SQL. Podaci u tablicama su dostupni preko kombinacije imena tablica, atributa (stupca) i jedinstvenog primarnog ključa. Podaci iz više različitih tablica mogu se spojiti i prikazati zajedno korištenjem stranog ključa, čime nastaje pogled načinjen od sjedinjenih tablica.

4.3.2. Implementacija baze podataka putem poslužitelja

Za uspješnu komunikaciju između poslužitelja i baze podataka, potrebno je ostvariti konekciju između njih. Pošto programski jezik JavaScript ne posjeduje metode i protokole za komunikaciju s bazom podataka, u tu svrhu korištena je biblioteka Knex.js koja omogućava traženu komunikaciju s više različitih vrsta relacijskih baza. Za početak rada potrebno je stvoriti vezu na način prikazan kodom 4.12 gdje se navode potrebni podaci za spajanje s bazom.

```
const database = knex({
    client: 'pg',
    connection: {
        host: '127.0.0.1',
        user: 'postgres',
        password: 'user-defined-password',
        database: 'zavrsni-rad-db'
    }
});
```

Programski kod 4.12. Spajanje poslužitelja s bazom podataka

4.3.3. Programsko rješenje poslužitelja

Kako je ranije navedeno, poslužitelj ima ključnu ulogu u povezivanju korisnika s bazom podataka radi pružanja potrebnih informacija korisnicima. Osim toga, u ovoj aplikaciji se brine o komunikaciji između dvije vrste korisničkih računa tijekom postupka otkrivanja pogreške ili kvara sve do njihovog uklanjanja. Poslužitelj od korisnika zaprima različite zahtjeve putem HTTP protokola, te zatim prema parametrima i metodi zahtjeva stvara odgovor koji korisnik dobije.

Prvi zahtjev prilikom svakog korištenja aplikacije odnosi se na prijavu korisnika u sustav, bilo da se radi o prijavi postojećeg korisnika, ili registraciji novog. Uspješno rukovanje ovim zahtjevom ostvareno je HTTP metodom tipa *POST*. Ova metoda se preporučeno koristi prilikom unosa novih podataka u bazu ili slanje raznih formi s korisnički unesenim podacima te je njeno korištenje prikazano na primjeru programskog koda 4.13. Pokušajem prijave korisnika, šalju se podaci unutar tijela zahtjeva te poslužitelj u slučaju ispravne prijave/registracije vraća odgovor koji korisniku omogućuje daljnje korištenje aplikacije.

```
app.post('/register', (request, response) => {
    const {email, name, password, vcode, checkboxState} = request.body;
    if(checkboxState === "true" && vcode !=="authorized")
        response.status(400).json("Wrong mechanic validation code.")
    else
        database.insert({
            password: password,
            email: email,
            name: name,
            joined: new Date(),
            type: (vcode === 'authorized' ? 'mech' : 'driv')
        })
        .into('users')
        .returning('*')
        .then( res => {
            response.json(res[0])
        })
        .catch(err => response.status(400).json(err))
})
```

Programski kod 4.13. Metoda za rukovanje registracijom novog korisnika

Osim navedenih primjera prijave i registracije korisnika, metoda tipa *POST* koristi se i za stvaranje novih zahtjeva za pregled te unos novih vozila u bazu. Iz navedenih primjera može se primijetiti kako je ova metoda karakteristična kada se zahtjev odnosi na unos novih redaka u bazu podataka aplikacije.

Po uspješnoj prijavi korisnika u sustav, potrebno je dohvatiti podatke od poslužitelja potrebne za rad korisnika u aplikaciji. U tu svrhu upotrebljava se metoda tipa *GET*. Ovaj tip metoda namijenjen je dohvaćanju podataka bez mijenjanja stanja na poslužitelju, te ga ne bi trebalo koristiti u druge svrhe. Prilikom slanja zahtjeva poslužitelju, metoda tipa *GET* prima parametre zahtjeva kroz zaglavlje ili adresu (*URL – Uniform Resource Locator*) zahtjeva kako je prikazano kodom 4.14.

U ovome slučaju, aplikacija šalje zahtjev kako bi pristupila svim zatraženim ili već zakazanim sastancima. Ovaj zahtjev je bitan dio prilikom izvedbe komunikacije između vozača i servisera, jer omogućava serviseru pregled traženih zahtjeva od strane vozača. Isto tako u suprotnom smjeru vozač pomoću podataka koje zaprima metodom tipa *GET* može vidjeti odgovor servisera na traženi zahtjev. Ova metoda se unutar aplikacije koristi još i za dohvaćanje svih vozila nekog vlasnika iz baze.

```
app.get('/previous-appointments/:id/:pendingreq', (req,res) =>{
    const {id} = req.params;
    const pending_req=req.params.pendingreq;

    database.select('*').from('cars')
    .join('appointments', 'appointments.serial_number','cars.serial_number')
    .select('*')
    .where({owner_id:id}).andWhere({pending_request: pending_req})
    .orderBy('scheduled_time')
    .then(data => {
        res.json(data)
    })
    .catch(err=> res.status(400).json(err))
})
```

Programski kod 4.14. Metoda za prikaz traženih termina i dijagnostika servisera

Prilikom potvrde termina dijagnostičkog zahvata od strane servisera, zahtjev mijenja stanje, odnosno u bazi podataka se mijenja atribut *pending_request* koji označava čeka li zahtjev odgovor od servisera. Kako bi se stanje unutar baze promijenilo, korištena je metoda tipa *PUT*. Ova metoda preporučena je upravo za izmjenu podataka unutar baze, te se kao i metodi tipa *POST* šalju argumenti unutar tijela. Osim za navedeni slučaj, metode tipa *PUT* korištene su i prilikom dijagnosticiranja kvara po unosu dijagnostičkog koda, narudžbu zamjenskog dijela i dovršavanje servisa te su dvije posljednje navedene metode prikazane programskim kodom 4.15.

```
app.put(`/end-service`, (req,res) =>{
    let {appointment_number, service_note} = req.body;
    database('appointments').where('appointment_number','=',appointment_number)
        .update({resolved: 'Y', service_note: service_note})
    .returning('*')
    .then(data => res.json(data[0]))
})

app.put(`/order-part`,(req,res)=>{
    let {appointment_number, ean} = req.body;
    database('appointments').where('appointment_number','=',appointment_number)
        .update({made_order : 'Y', ean: ean})
        .returning('*')
    .then(data => res.json(data[0]))
})
```

Programski kod 4.15. Metode *PUT* za dovršetak servisa i narudžbu zamjenskog dijela

Posljednje korišteni tip metode za rukovanje HTTP zahtjevima je metoda *DELETE*. Ovaj tip metode služi za uklanjanje redaka iz baze podataka, a unutar aplikacije koristi se na jednom mjestu, prilikom odbijanja traženog zahtjeva dijagnostike na strani servisera. Rukovanje tim zahtjevom prikazano je programskim kodom 4.16

```
app.delete('/reject-appointment/:id', (req,res) =>{
    let {id} = req.params;
    database('appointments').where('appointment_number', '=', id)
    .del()
    .then(data => res.json(data))
})
```

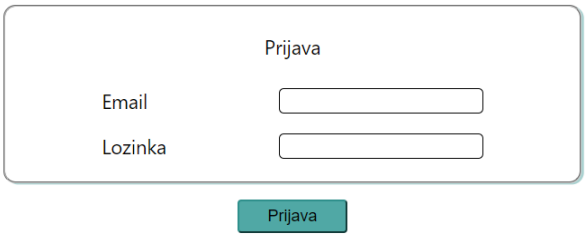
Programski kod 4.16. Metoda za odbijanje traženog zahtjeva dijagnostike

5. KORIŠTENJE I ISPITIVANJE APLIKACIJE S ANALIZOM REZULTATA

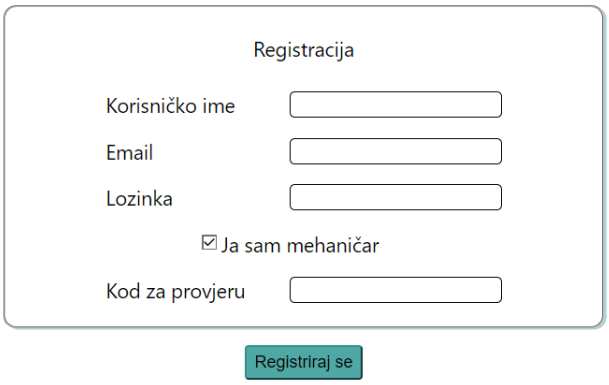
Unutar ovog poglavlja prikazano je korištenje aplikacije i njenih mogućnosti kroz implementirane funkcionalnosti, od početka korištenja do svih koraka tijekom procesa servisa te prikaza završenog postupka na kraju servisa.

5.1. NAČIN KORIŠTENJA WEB APLIKACIJE

Za početak korištenja aplikacije, korisnik se treba prijaviti ili registrirati u aplikaciju ispunjavanjem jedne od formi (slika 5.1 i 5.2).



Slika 5.1. Forma prijave

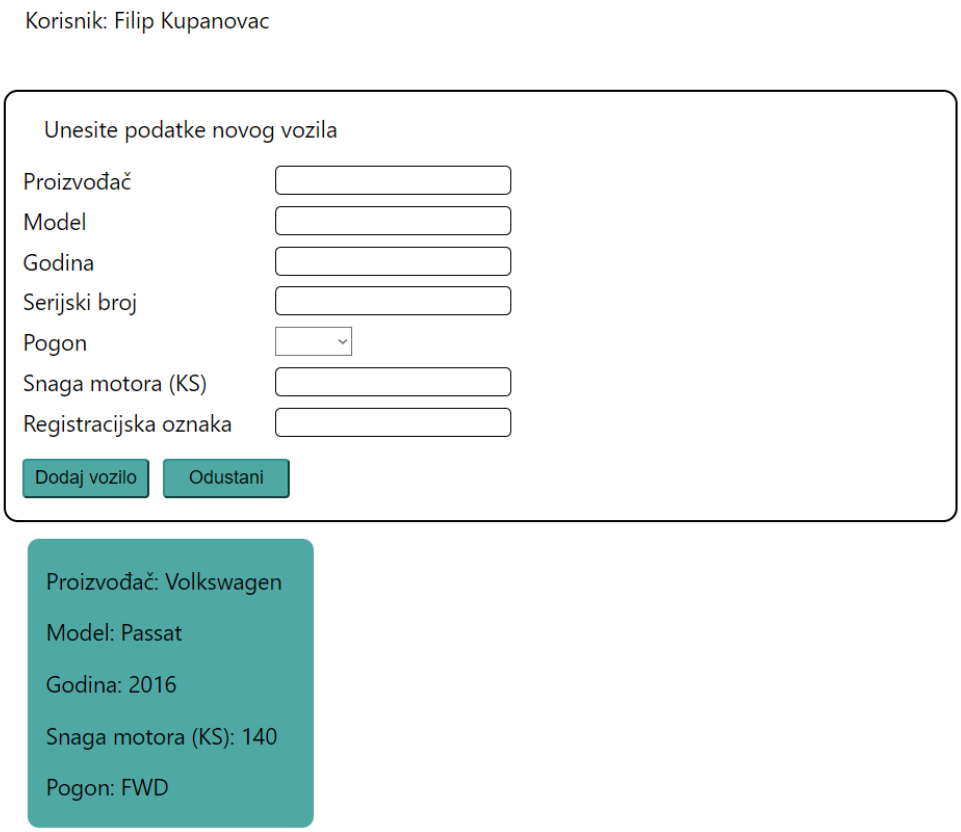


Slika 5.2. Forma registracije

Po uspješnoj prijavi ili registraciji, otvara se sučelje karakteristično toj vrsti korisničkog profila.

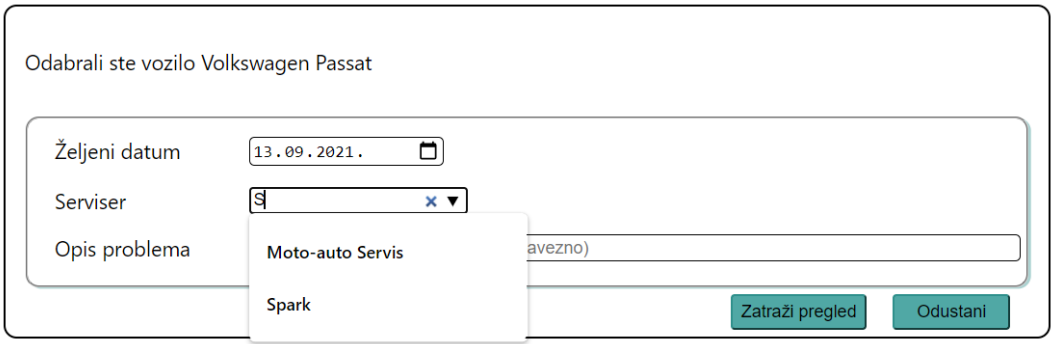
5.1.1. Način rada vozača

Prijavom vozača na sučelju aplikacije prikazuje se korisničko ime vozača u gornjem lijevom kutu te gumb za dodavanje vozila u gornjem desnom kutu. Iz baze podataka aplikacija dohvaća sva vozila prijavljenog korisnika te ih prikazuje u obliku kartica s podacima o vozilu. Klikom na gumb *Dodaj vozilo* otvara se forma za unos podataka o novom vozilu prikazana na slici 5.3. Po ispravnom unosu svih traženih podataka, korisniku se prikazuje vozilo unutar kontejnera za kartice vozila.



Slika 5.3. Forma za unos vozila i kontejner kartica vozila

Klikom na karticu vozila, ono postaje aktivno ako nije bilo ranije. Ukoliko je bilo aktivno, ponovnim klikom se poništava stanje aktivnog vozila. Za aktivno vozilo tada je moguće tražiti pregled kod željenog servisera. Unutar okvira koji se pojavljuje nalazi se gumb *Traži novi pregled* te klikom na njega otvara se forma za stvaranje novog zahtjeva (slika 5.4). Forma traži obavezan unos željenog datuma i servisera kod kojega bi vozač htio obaviti dijagnostički pregled. Neobavezna mogućnost uz to je da vozač sam opiše simptome radi kojih želi obaviti pregled. Radi lakšeg odabira željenog servisera, tijekom unosa u polje za odabir servisera dinamički se mijenja padajući izbornik svih servisa koji sadrže unesenu vrijednost u svojem imenu. Po unosu svih obaveznih stavki, omogućava se da klikom na gumb *Zatraži pregled* stvori novi zahtjev koji je potom prikazan unutar okvira sa svim traženim zahtjevima za pregled (slika 5.5).

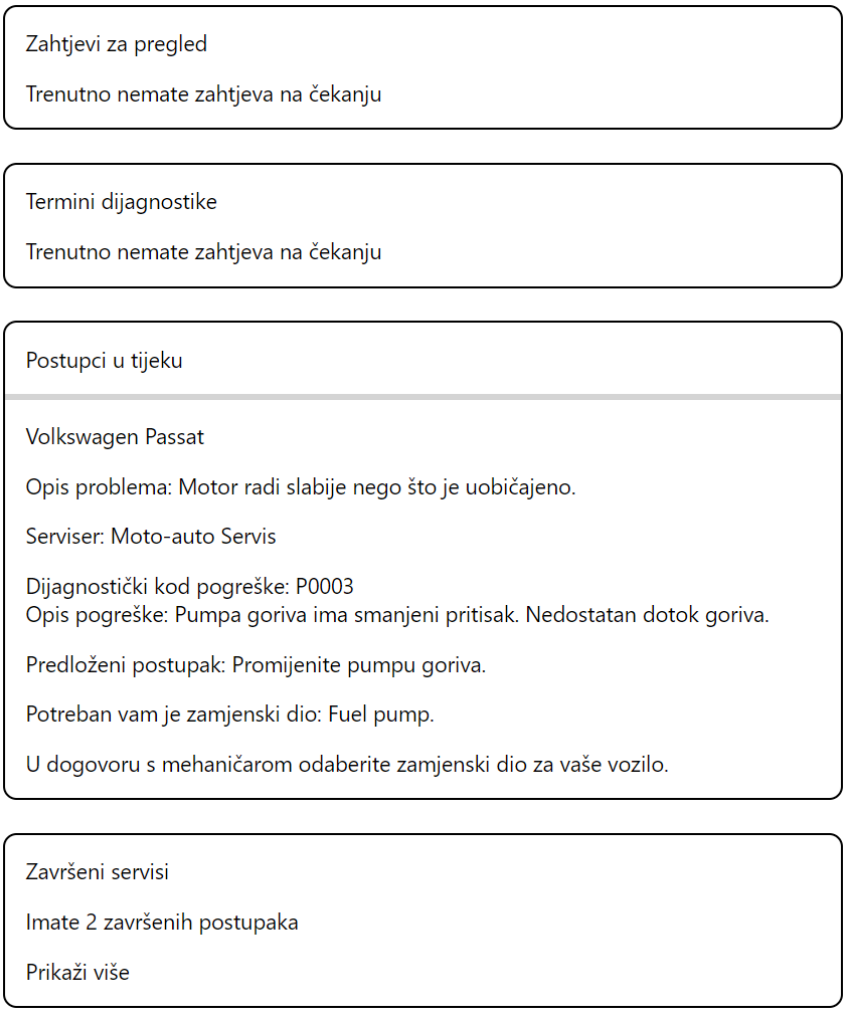


Slika 5.4. Forma za stvaranje novog zahtjeva s padajućim izbornikom

Svi zahtjevi korisnika su unutar aplikacije podijeljeni u 4 skupine, od kojih svaka ima svoj kontejner kojima se postiže njihova vizualna podjela radi lakšeg snalaženja korisnika. Navedene skupine su:

- Zahtjevi za pregled – Korisnik je zatražio pregled kod servisera
- Termini dijagnostike – Serviser potvrdio pregled uz moguću izmjenu datuma
- Postupci u tijeku – Završena dijagnostika, čeka se narudžba i dostava dijela ukoliko je potreban popravak
- Završeni servisi – Prikazane informacije o obavljenim servisima

Svaka od kartica unutar navedenih kontejnera prikazuje informacije potrebne i poznate u tom koraku servisnog postupka. Tako se tek po obavljenoj dijagnostici prikazuje kvar na vozilu dok se zatim još dodatno pojavljuje informacija o potrebnoj vrsti dijela, ili specifičnom naručenom dijelu ovisno o tome je li serviser u dogovoru s korisnikom naručio dio prema potrebama i mogućnostima korisnika (slika 5.5).



Slika 5.5. Kontejneri s fazama servisa

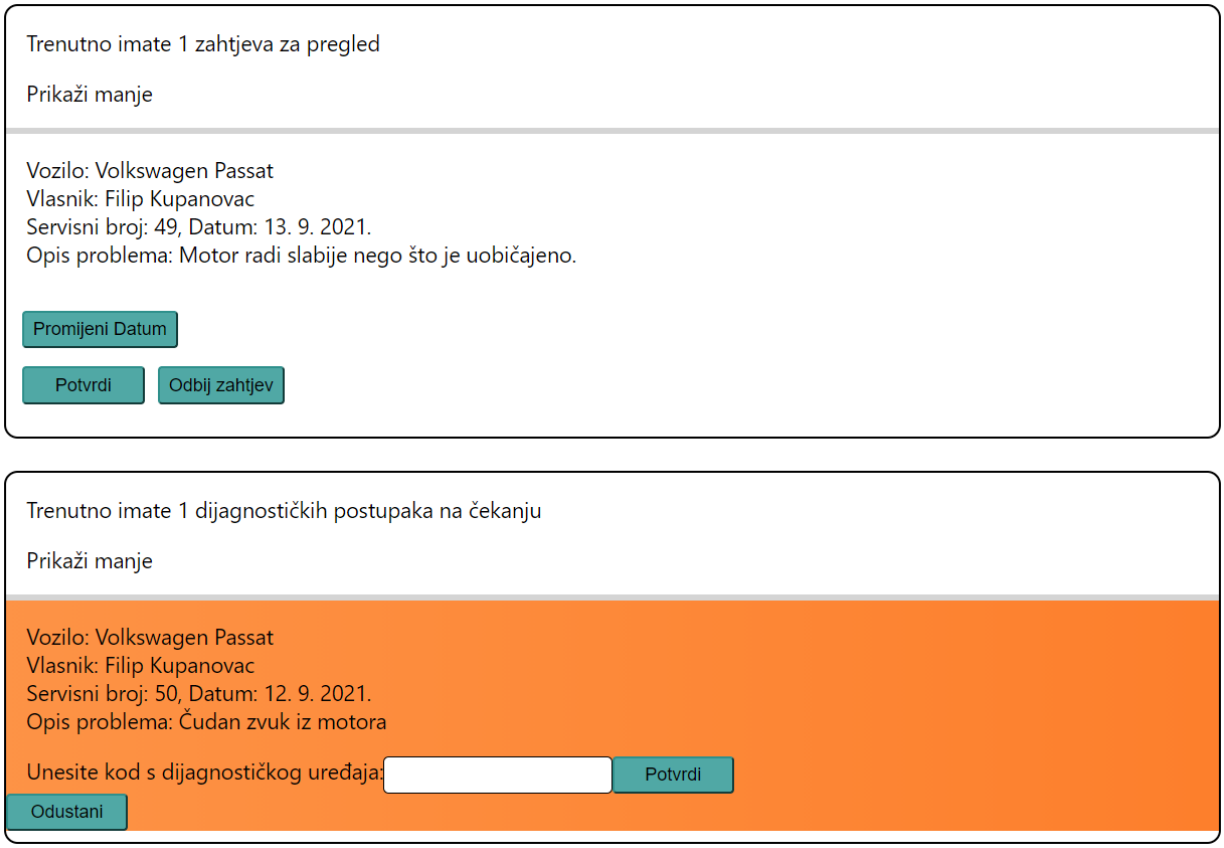
5.1.2. Način rada servisera

Jednako kao i kod vozača, kod servisera se klikom na gumb *Dodaj novo vozilo* nudi mogućnost dodavanja vozila u bazu. Jedina razlika je što u ovom slučaju serviser to može učiniti samo za drugog vozača. Radi toga je njegova forma za popunjavanje podataka proširena s dodatnim poljem za unos identifikatora vozača, polje za unos pod nazivom *ID Vlasnika*.

Nakon toga, poput vozača ima također i četiri skupine postupaka podijeljenih prema istom načelu navedenom ranije u odjeljku 5.1.1, uz dodane funkcionalnosti kako bi postupak prelazio u sljedeću fazu. Prva razlika je u tome što može proširiti i sažeti prikaz postupaka po volji klikom na tekst *Prikaži više*, odnosno *Prikaži manje*, ovisno o trenutnom stanju prikazu kako bi se mogao fokusirati na ono što mu je u tom trenutku od veće važnosti. Svaki od postupaka nudi informacije o vozilu i vlasniku istog, te se prikazuje vozačev osvrt na uočeni problem ukoliko ga je naveo. Za bolju uočljivost važnijih postupaka, boja pozadine pojedinog termina se mijenja u narančastu

iskazujući tako upozorenje za sve postupke zakazane na trenutni dan, kako je prikazano na slici 5.6.

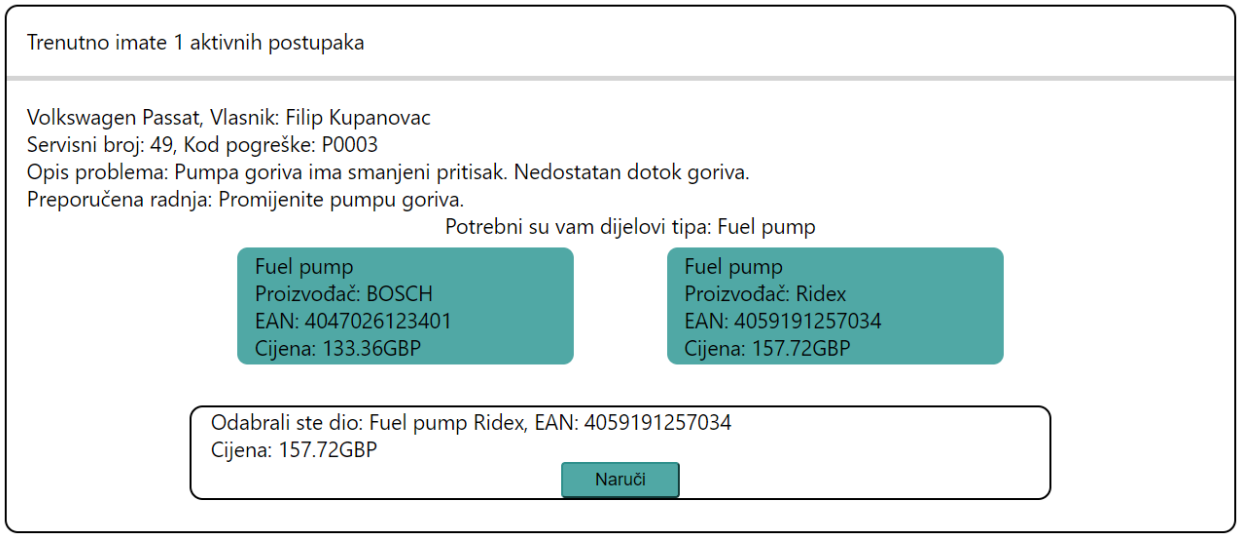
Klikom na neku od kartica traženog dijagnostičkog termina ili već dogovorenog dijagnostičkog postupka, on postaje aktivan te je moguće odraditi sljedeći korak prilikom servisa. Gumb *Promijeni Datum* nudi mogućnost promjene traženog datuma serviseru ukoliko mu korisnički zatražen datum ne odgovara. Klikom na taj gumb prikazuje se polje za unos tipa datum, čija se vrijednost zatim uzima kao potvrđeni datum dijagnostičkog zahvata. Ispod toga, nalaze se dva gumba. Gumb *Potvrdi* nudi mogućnost potvrde dijagnostičkog termina te tada postupak prelazi u sljedeću fazu, dok se klikom na gumb *Odbij* traženi zahtjev briše.



Slika 5.6. Traženi i potvrđeni aktivni dijagnostički termin

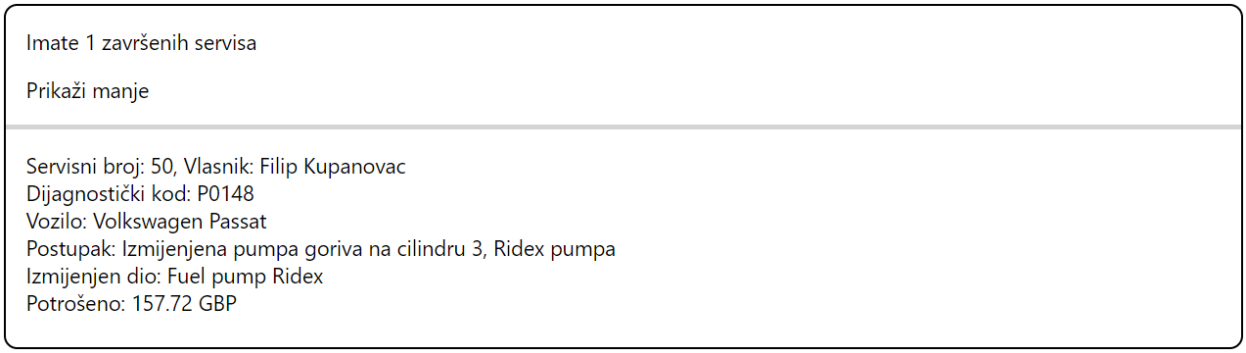
Tijekom dijagnostičkog postupka serviser na uređaju za dijagnostiku očitava kod pogreške. Zatim unutar aplikacije klikom na gumb *Započni dijagnostiku* otvara se polje za unos dijagnostičkog koda pogreške (prikazano na slici 5.6). Potvrdom kod na klik gumba *Potvrdi* pretražuje se baza dijagnostičkih kodova te ukoliko je unesen ispravan kod (koji se nalazi u bazi) postupak prelazi u sljedeću fazu, analizu dijagnosticiranog problema i narudžbu dijela ukoliko je potreban zamjenski dio.

Aktivni postupci u načinu rada servisera se razlikuju od svih ostalih faza po tome što ih nije moguće sažeti, odnosno maknuti sa sučelja jer je na njih stavljena najveća važnost. Od informacija sadrži podatke o vozilu i dijagnosticiranoj pogrešci, kao i preporučeni zahvat. Unutar kartice još su ponuđeni dijelovi za zamjenu kompatibilni s vozilom. Kada se odabere jedan od ponuđenih dijelova, on postaje aktivan te se ispod popisa dijelova nudi mogućnost narudžbe odabranog dijela (slika 5.7). Po narudžbi dijela, prikazan je okvir s informacijom o naručenom dijelu.



Slika 5.7. Postupak u tijeku

Po primitku naručenog dijela, serviser je u mogućnosti zamijeniti dio i obaviti servis kad god vozač doveze automobil na servis. U aplikaciji je po dovršenom servisu serviser dužan unijeti izvještaj o servisu gdje opisuje postupak izmjene te klikom na gumb *Završi servis* se zaključuje odabrani servisni postupak. Naposljetku svi do kraja odrađeni servisni postupci bivaju prikazani unutar kartice *Završeni servisi* prikazanoj slikom 5.8.



Slika 5.8. Dovršeni postupci servisera

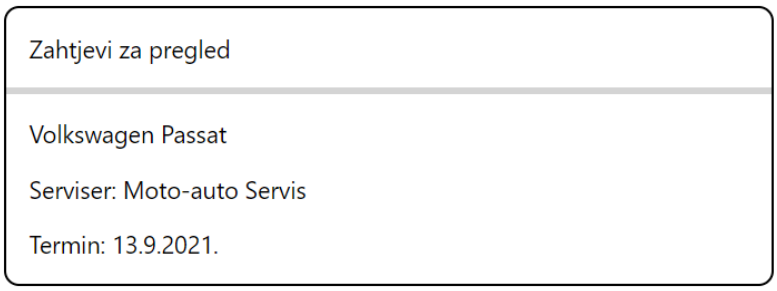
5.2. ISPITIVANJE RADA APLIKACIJE

U ovome dijelu biti će prikazan rad aplikacije kroz nekoliko korisničkih slučajeva kako bi se ispitalo korištenje aplikacije i utvrdila njena ispravnost. Kroz nekoliko korisničkih profila i slučaja korištenja ispitati će se funkcionalnosti aplikacije zajedno s popratnom analizom rezultata.

5.2.1. Korisnički slučaj 1

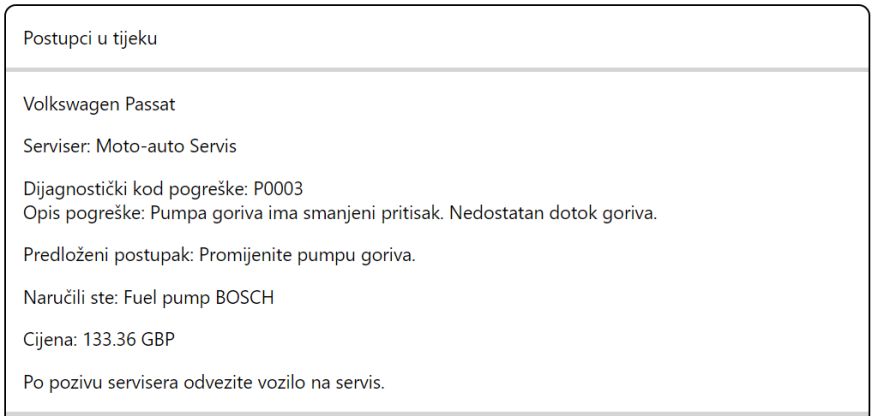
Za ovaj slučaj uzet je postojeći korisnik s profilom vozača. Po uočenom neuobičajenom ponašanju automobila odlučuje se potražiti dijagnostički pregled prije nego na vozilu nastane kvar i ispunjava zahtjev putem forme prikazane na slici 5.4. sa sljedećim parametrima: vozilo – VW Passat, željeni datum – 13.09.2021., serviser: Moto-auto Servis te ne navodi vlastiti opis problema.

Nakon ispunjavanja zahtjeva, isti je prikazan vozaču kako je prikazano na slici 5.9 u okviru traženih zahtjeva. Po potvrdi servisera bez izmjene datuma, zahtjev se jednako prikazuje u okviru dijagnostičkih termina te je jedina razlika što je tada termin dogovoren.



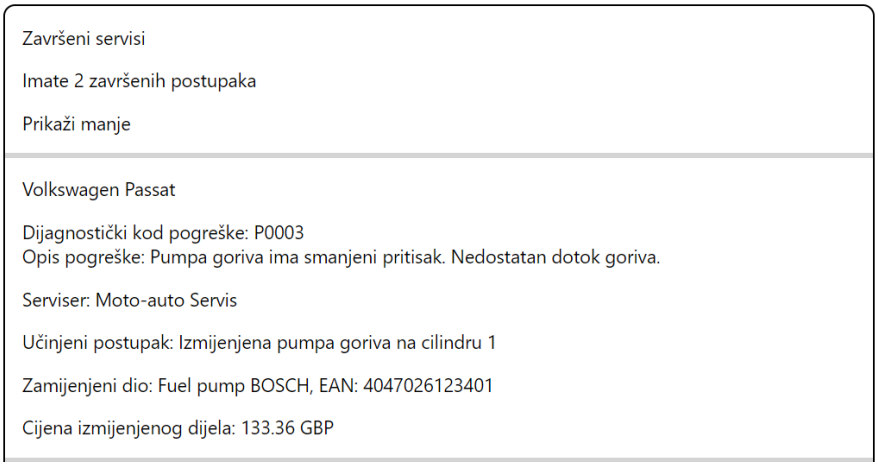
Slika 5.9. Prikaz zahtjeva za pregled

Dijagnostičkim postupkom utvrđena je pogreška na pumpi goriva cilindra 1 te u dogovoru sa serviserom vozač odabire novu pumpu goriva za izmjenu. Potom mu nakon narudžbe unutar okvira postupaka u tijeku (slika 5.10) pišu informacije o dijagnostičkom postupku i naručenom dijelu.



Slika 5.10. Postupak u tijeku s naručenim zamjenskim dijelom

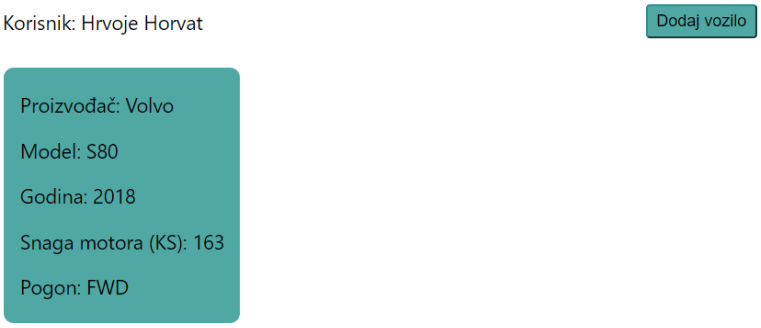
Po završetku servisnog postupka, vozaču je prikazana informacija o izmijenjenom dijelu uz opis servisera o učinjenom postupku unutar okvira završenih servisa zajedno sa ostalim prethodnim servisima za sva vozila tog vozača (slika 5.11).



Slika 5.11. Izvještaj završenog servisa

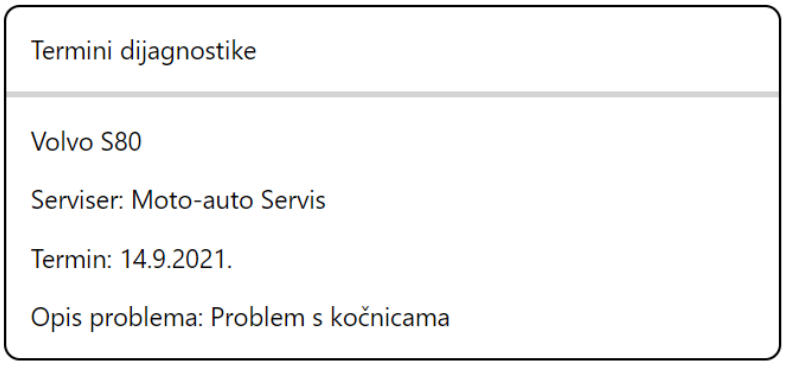
5.2.2. Korisnički slučaj 2

Korisnik 2 predstavlja novog korisnika aplikacije prijavljenog kao vozač, koji po prijavi nema niti jedno vlastito vozilo u bazi podataka te traži servisera pomoć oko unosa vlastitog vozila što je prikazano u odjeljku 5.2.3. Po unosu vozila u bazu prikazuje mu se vozilo kao na slici 5.12 s pripadnim podacima. Zatim potražuje servis na datum 13.09.2021. u servisu Moto-auto Servis s opisom problema: „Problem s kočnicama“.



Slika 5.12. Prikaz unesenog novog vozila od strane servisera

Obzirom da serviser nije u mogućnosti zaprimiti vozilo na zahtijevani datum, pomiče dijagnostički postupak za sljedeći dan, što korisnik vidi prikazano u dogovorenim terminima dijagnostike kao na slici 5.13.

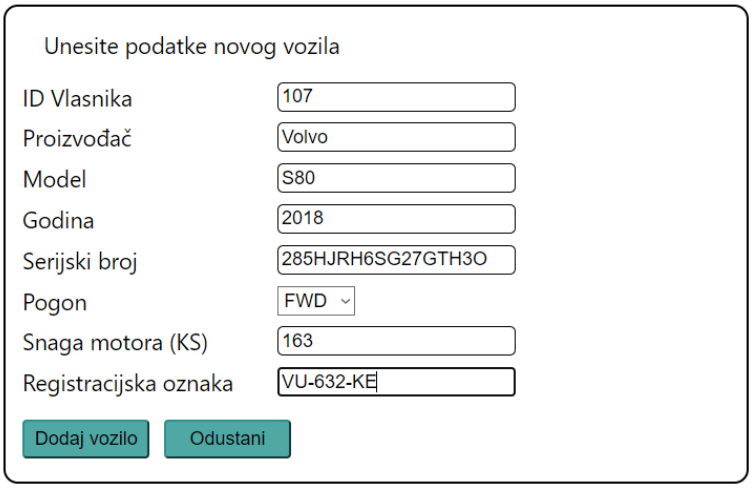


Slika 5.13. Dogovoreni termin dijagnostike

Dijagnostičkim postupkom utvrđena je pogreška s kodom *C0127 - niska razina ulja za kočnice*, radi čega su kočnice imale lošije performanse. Naručeno je jedno od ulja koje odgovara marki automobila Volvo te po pristizanju narudžbe i izvršavanju servisa, postupak je završen, a vozač može vidjeti završeni postupak unutar odgovarajućeg okvira.

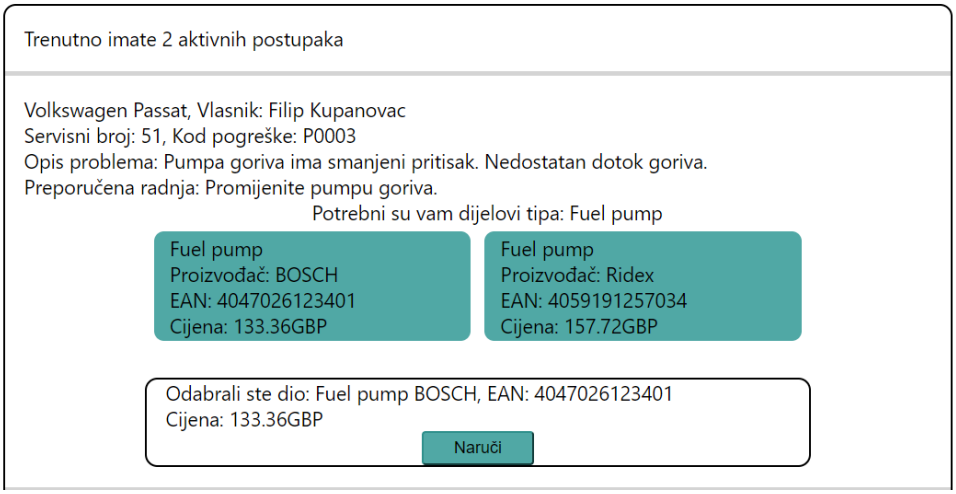
5.2.3. Korisnički slučaj 3

Korisnik 3 predstavlja Moto-auto Servis koji je naveden u prethodnim odjeljcima ispitivanja rada kao serviser. Serviser u dogovoru s vlasnikom vozila, za vlasnika ispunjava formu za unos vozila u bazu, prikazanu slikom 5.14. Potom mu pristižu dva zahtjeva za dijagnostičkim postupkom, s istim traženim datumom. Pošto nije u mogućnosti tog dana učiniti oba pregleda, potvrđuje prvi, dok drugi odgađa za sljedeći dan klikom na gumb *Promijeni Datum*, unosom novog datuma te na posljetku klikom gumba *Potvrdi*.



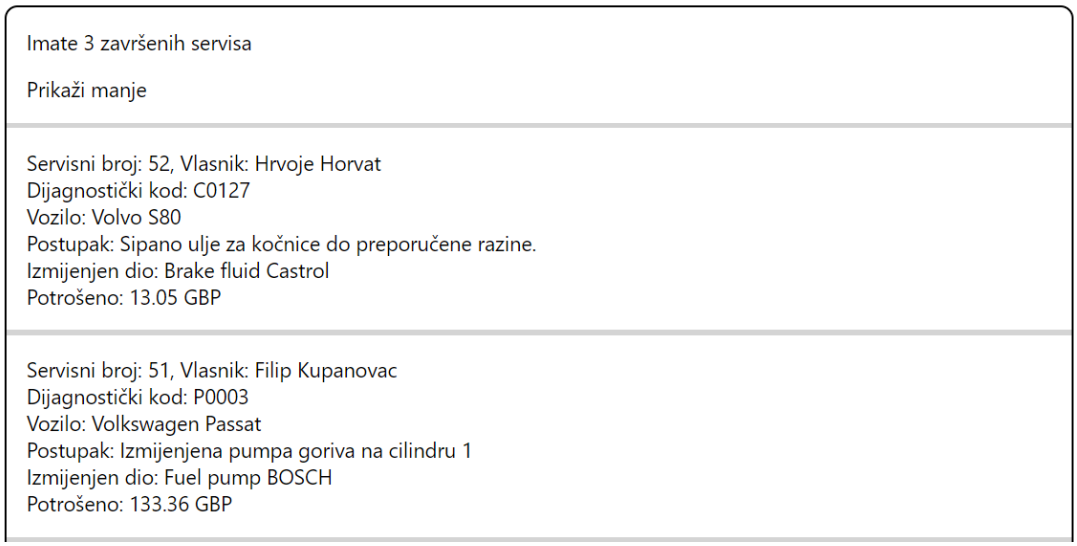
Slika 5.14. Serviser unosi podatke novog vozila za vozača

Dijagnostičkim pregledom prvog zahvata utvrđen je kod pogreške *P0003 - Pumpa goriva ima smanjeni pritisak. Nedostatan dotok goriva*. Unosom koda pogreške u aplikaciju, postupak prelazi u stanje postupka u tijeku te su prikazani zamjenski dijelovi koji odgovaraju vozilu s pogreškom. Klikom na jedan od dijelova otvara se mogućnost narudžbe klikom gumba *Naruči* (na slici 5.15).



Slika 5.15. Prikaz postupka u tijeku s narudžbom dijela

Po završetku oba postupka, isti se prikazuju serviseru za evidenciju zajedno sa svim ostalim završenim servisima kako je prikazano slikom 5.16.



Slika 5.16. Prikaz dovršenih servisa

5.3. ANALIZA RADA APLIKACIJE

Aplikacija korisnicima pruža iznimno jednostavno korisničko sučelje uz težnju k minimalističkom dizajnu radi postizanja bržeg učitavanja stranice i odziva. Svaki korak prilikom korištenja aplikacije sadrži opis što korisnik treba učiniti ukoliko ima mogućnost izvršavanja neke funkcionalnosti. Dodavanje novih korisnika je iznimno jednostavno, kao i automobila u vlasništvu vozača. Ispitivanjem aplikacije u potpoglavlju 5.2 prikazane su primjerom funkcionalnosti aplikacije te je utvrđeno kako sve funkcionira prema ranije navedenom dijagramu toka aplikacije bez neočekivanih rezultata, što je potkrijepljeno priloženim slikama tijekom postupka ispitivanja aplikacije.

Korištenjem aplikacije postignuto je analiziranje stanja vozila prije nastanka kvara te su tako vozilima na vrijeme uočene nastale pogreške. Narudžbom dijela s popisa dijelova koji odgovaraju tom automobilu, naručen je zamjenski dio dok je vozilo bilo u voznom stanju te je po primitku dostave zamjenskog dijela učinjen servis kako bi se otklonile pogreške, a vozilo kroz cijeli taj period bilo u voznom stanju, odnosno na raspolaganju vlasniku. Na taj način je uspješno provedena učinkovita nabava rezervnog dijela.

6. ZAKLJUČAK

Kako su kvarovi na svakom automobilu jedna od neizbježnih pojava s kojima se susreće svaki vozač, veliko olakšanje vozaču predstavljalo bi otkrivanje potencijalnog kvara i zamjena rezervnog dijela prije nastanka kvara ili što prije nakon nastanka kvara. S tim ciljem nastala je i web aplikacija prikazana u radu. Uz korištenje protokola OBD II za dijagnostiku kvarova, analizira se stanje vozila i otkriva pogreška, odnosno dio koji treba zamijeniti. Aplikacija pronalazi zamjenske dijelove na temelju analize stanja vozila te serviser u dogovoru s vozačem nabavlja odgovarajući zamjenski dio. Time se ubrzava postupak otklanjanja pogreške i komunikacije između servisera i potrošača što dovodi do krajnjeg cilja aplikacije, a to je sprječavanja kvara prije njegovog nastanka, odnosno pravovremene nabave rezervnih dijelova kako bi automobil bio što kraći vremenski period u neispravnom stanju.

Web aplikacija razvijena je pomoću integriranog razvojnog okruženja Microsoft Visual Studio Code. Prilikom razvoja aplikacije korišten je CSS za uređivanje sadržaja aplikacije i programski jezik JavaScript za definiranje funkcionalnosti unutar aplikacije. Aplikacija je ispitana kroz tri korisnička slučaja te je analizom primjera prikazano kako je aplikacija ispravna i funkcionalna, uz zadovoljavanje postavljenih zahtjeva. Moguća poboljšanja aplikacije su detaljnija podjela automobila prema tipu ugrađenog motora, praćenje već korištenih dijelova prilikom servisa za grupu istovrsnih vozila uz ostavljanje povratnih informacija i ocjene dijela kako bi serviser mogao predložiti vozaču onaj dio koji najbolje odgovara njegovom vozilu na temelju iskustava svih korisnika aplikacije. Također, moguće je vizualno doraditi korisničko sučelje čime bi se aplikacija učinila vizualno privlačnijom za korištenje u svrhu ostvarenja boljeg korisničkog iskustva.

LITERATURA

[1] Dr. sc. Amir Halep, dipl. inž, dr. sc. Ranko Antunović, dipl. inž., »Defekt, kvar i otkaz,« u *4. Konferencija "ODRŽAVANJE 2016"*, Zenica, 2016.

[2] B. Jardini, M. El Kyal i M. Amri, »The management of the supply chain by the JIT system (Just In Time) and the EDI technology (Electronic Data Interchange),« u *3rd International Conference on Logistics Operations Management (GOL)*, Fez, Morocco, 2016.

[3] CarProblemZoo.com, »Car Problems, Statistics, and Analysis,« 2021. [Mrežno]. Dostupno na: https://www.carproblemzoo.com/. [Pristupljeno: 16 svibanj 2021.].

[4] J. D. Power, »Vehicle Dependability at All-Time High, J.D. Power Finds,« 2021.

[5] M. D. Forecast, »Europe Automotive Aftermarket Market by Product, by Distribution Channel, by Application - Regional Analysis and Industry Forecast, 2021-2026,« travanj 2021.. [Mrežno]. Dostupno na: https://www.marketdataforecast.com/market-reports/europe-automotive-aftermarket-market. [Pristupljeno: 16 svibanj 2021.].

[6] B. Frowein, N. Lang, F. Schmieg i G. Sticher, Returning to Growth - A Look at the European Automotive Aftermarket, 2014: The Boston Consulting Group (BCG).

[7] »Online car parts: Buy cheap auto parts and spares online,« [Mrežno]. Dostupno na: https://www.onlinecarparts.co.uk. [Pristupljeno: 3 lipanj 2021.].

[8] » EUspares Online Shop,« [Mrežno]. Dostupno na: https://www.euspares.co.uk. [Pristupljeno: 3 lipanj 2021].

[9] »Guide to non-functional requirement: types and examples,« [Mrežno]. Dostupno na: https://www.boxuk.com/insight/guide-to-non-functional-requirements-types-and-examples/. [Pristupljeno: 7 rujan 2021].

[10] Z. Szalay, Z. Kánya, L. Lengyel, P. Ekler, T. Ujj, T. Balogh i H. Charaf, »ICT in road vehicles — Reliable vehicle sensor information from OBD versus CAN,« u *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, Budapest, 2015.

[11] SAE International Website, »J1979_201202: E/E Diagnostic Test Modes,« 23 veljača 2012.. [Mrežno]. Dostupno na: https://www.sae.org/standards/content/j1979_201202/. [Pristupljeno: 25 kolovoz 2021.].

[12] D. Rimpas, A. Papadakis i M. Samarakou, »OBD-II sensor diagnostics for monitoring vehicle operation andconsumption,« ScienceDirect, Athens, Greece, 2019.

[13] »OBD2 standard fault codes - Full list - Outils OBD Facile,« Outils OBD Facile SAS, 2018. [Mrežno]. Dostupno na: https://www.outilsobdfacile.com/data-trouble-code-obd2.php. [Pristupljeno: 28 kolovoz 2021].

[14] »HTML: HyperText Markup Language,« MDN Web Docs, 8 srpanj 2021.. [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTML. [Pristupljeno: 4 srpanj 2021.].

[15] T. Barners-Lee, »Information Management: A Proposal,« CERN, 1989./1990..

[16] E. Mujadžević, Uvod u CSS: C220, C220-20150219 ur., G. Kurtović, Ur., Zagreb: Sveučilište u Zagrebu Sveučilišni računski centar (Srce), 2014.

[17] B. Bos, »A brief history of CSS until 2016,« u *Cascading Style Sheet – designing for the Web*, 3. ur., Addison-Wesley Professional, 2016..

[18] M. Haverbeke, A modern Introduction to Programming - Eloquent JavaScript, 3. ur., No Starch Press, 2018.

[19] Nodejs.dev, »Introduction to Node.js,« 2021.. [Mrežno]. Dostupno na: https://nodejs.dev/learn. [Pristupljeno: 30 kolovoz 2021.].

[20] R. Manger, Baze podataka (Database Systems), Zagreb: Element, 2012..

ŽIVOTOPIS

Filip Kupanovac rođen je 18. lipnja 1999. godine u Osijeku. Nakon završetka svojeg osnovnoškolskog obrazovanja u Osnovnoj školi Ivana Kukuljevića u Belišću, upisuje Srednju školu Valpovo u Valpovu, smjer Elektrotehničar. Tijekom svojeg srednjoškolskog obrazovanja sudjelovao je četiri puta na školskom i županijskom natjecanju iz matematike i fizike, jednom na državnom natjecanju iz osnova elektrotehnike te tri puta na državnom natjecanju iz informatike na kojem je ostvario zavidno drugo mjesto. Osim toga, u veljači 2018. godine imao je priliku sudjelovati u mobilnosti učenika unutar Erasmus projekta pod nazivom „Novim tehnologijama na tržište rada“ u španjolskoj Sevilli. Iste godine nakon završetka srednje škole sa odličnim uspjehom upisuje preddiplomski studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Uz obaveze studiranja, veliku pažnju još od 2016. godine pridodaje svojoj karijeri košarkaškog suca u kojoj je 2021. godine došao do titule državnog suca, najvišeg ranga košarkaškog suca u Republici Hrvatskoj. Uz to, u slobodno vrijeme pridodaje i veliku pažnju tečajevima namijenjenim za usavršavanje programskih jezika koji su mu, uz redovno učenje na fakultetu, omogućili da savlada i dodatno usavrši programske jezike C, C#, Java te JavaScript. Unutar svojih jezičnih vještina, dosegao je zavidnu razinu znanja engleskog jezika kojim se bez poteškoća može koristiti u društvene i profesionalne svrhe, a uz njega poznaje španjolski i njemački.

_____

Potpis autora

SAŽETAK

U ovom završnom radu osmišljena je i programski ostvarena web aplikacija za poboljšanje učinkovitosti nabave rezervnih dijelova na temelju analize kvarova. Aplikacija je razvijena koristeći Visual Studio Code, dok je programski kod pisan u programskom jeziku JavaScript. Aplikacija omogućuje analizu oštećenja korištenjem protokola On-board diagnostic II, te povezivanjem s *on-line* tržištem dijelova narudžbu zamjenskog dijela kako bi oštećenje bilo otklonjeno prije nastanka kvara ili u što kraćem roku nakon nastanka kvara, skraćujući tako proces izmjene dijela i čekanja na zamjenski dio. Ispitivanje web aplikacije za prikazane korisničke slučajeve prikazuje uspješno ostvarenje postupka nabave rezervnih dijelova prije nastanka kvara čime je postignuto poboljšanje učinkovitosti nabave i potvrđena ispravnost rada aplikacije.

Ključne riječi: automobilska industrija, kvarovi, nabava dijelova, servis vozila, web aplikacija.

ABSTRACT

Web application for procurement of spare car parts efficiency increase based on fault analysis

The goal of this bachelor's thesis is to develop Web application for procurement of spare car parts efficiency increase based on fault analysis. Application is developed using Visual Studio Code editor, and programming language JavaScript for implementing functionalities. Application allows usage of OBD-II protocol in fault analysis and connection to online spare parts market for needed spare part procurement before malfunction occurs, shortening thus procedure duration and waiting for spare part's arrival. Testing of application proved through 3 user cases successful realization of spare parts procurement before malfunction occured, by which was achieved efficiency increase, and confirmed correctness of the application.

Key words: automotive industry, faults, parts procurement, vehicle service, Web application

PRILOZI

Prilog 1. Završni rad u formatu docx

Prilog 2. Završni rad u formatu pdf

Prilog 3. Programski kod web aplikacije