Razvoj web aplikacije za prodaju automobila pomoću MERN Stack razvojnog okvira

Džin, Hana

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za prodaju automobila pomoću MERN Stack razvojnog okvira

Rijeka, srpanj 2024.

Hana Džin

0069089974

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za prodaju automobila pomoću MERN Stack razvojnog okvira

Mentor: Izv. prof. dr. sc. Marko Gulić

Rijeka, srpanj 2024.

Hana Džin

0069089974

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 21.03.2024.

Zavod: Zavod za računarstvo

Predmet: Razvoj web aplikacija

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Hana Džin (0069089974)**

Studij: Sveučilišni prijediplomski studij računarstva (1035)

Zadatak: **Razvoj web aplikacije za prodaju automobila pomoću MERN Stack razvojnog okvira / Development of a web application for car sales using the MERN Stack development framework**

Opis zadatka:

Razviti web aplikaciju za prodaju automobila. Aplikacija mora imati kompletnu funkcionalnost kupnje automobila uz implementirano plaćanje koristeći PayPal online servis za plaćanje. Nadalje, aplikacija mora imati funkcionalnost filtriranja automobila po određenim karakteristikama kao i mogućnost postavljanja recenzija onih korisnika koji su odradili kupnju preko web aplikacije. Također, treba implementirati cjelovitu autentifikaciju unutar spomenute web aplikacije uz odvojeni administracijski i korisnički dio aplikacije. Nadalje, treba implementirati razmjenu poruka između administratora i korisnika u realnom vremenu kao i mogućnost pretplate korisnika da na email prima sadržaj koji ga zanima. Aplikaciju treba izraditi upotrebom MERN Stack razvojnog okvira koji se sastoji od 4 tehnologije (MongoDB, Express, React, Node.js). Za vizualni dizajn web aplikacije koristiti Tailwind CSS radni okvir.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 22.03.2024.

Predsjednik povjerenstva za završni ispit:
prof. dr. sc. Miroslav Joler

Mentor:
doc. dr. sc. Marko Gulić

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad

Rijeka, srpanj 2024.

Džin Hana

Hana Džin

Zahvala

Veliko i iskreno hvala mojoj porodici – tati Jasminu, mami Meliti i bratu Harisu koji su mi kroz cijelo obrazovanje bili bezuslovna podrška.

SADRŽAJ

1 UVOD

AutoLoveRi web je aplikacija osmišljena za prodaju novih i rabljenih automobila. Ona služi kao virtualni izlog tvrtke specijalizirane za prodaju vozila, omogućujući potencijalnim kupcima da istraže ponudu automobila, pročitaju recenzije zadovoljnih korisnika i steknu uvid u način poslovanja iste. U širokoj ponudi automobila, korisnici mogu filtrirati i sortirati vozila kako bi se proces pronalaska željenog automobila olakšao i ubrzao, a po odabiru odgovarajućeg vozila kupac ga može kupiti koristeći PayPal online servis za plaćanje. Kako bi se omogućila razmjena podataka između administratora i krajnjeg korisnika u slučaju dodatnih pitanja ili provjere stanja narudžbe, aplikacija podržava i mogućnost dopisivanja ovih dviju strana u realnom vremenu.

Za izradu aplikacije korišten je MERN Stack razvojni okvir koji se sastoji od ukupno 4 tehnologije – MongoDB, Express, React i Node.js, a njen vizualni dizajn ostvaren je pomoću Tailwind CSS radnog okvira. Jedna od najvećih prednosti korištenja ovog razvojnog okvira leži u činjenici da sve njegove komponente koriste JavaScript, čime se omogućava dosljedan razvoj proces budući da se isti programski jezik koristi za pisanje koda na poslužiteljskoj i klijentskoj strani aplikacije.

Klijentska strana AutoLoveRi aplikacije izrađena je korištenjem Reacta, open-source JavaScript knjižnice razvijene od strane Facebooka, koja pojednostavljuje izgradnju interaktivnih korisničkih sučelja. Umjesto rada s cijelim sučeljem kao jednom nerazdvojivom cjelinom, React omogućava razdvajanje na pojedinačne, višekratno upotrebljive komponente. Ovakav pristup ubrzava renderiranje stranica i olakšava stvaranje dinamičnih web aplikacija koje se lako prilagođavaju potrebama korisnika.

Za spremanje podataka odabran je MongoDB, popularan open-source sustav za upravljanje bazom podataka (engl. DBMS - *Database Management System*) koji je poznat po svojoj fleksibilnosti i sposobnosti brzog rukovanja različitim vrstama podataka. Umjesto klasičnih tablica i redaka, MongoDB koristi dokumente koji omogućuju jednostavnije upravljanje i lakše prilagodbe podataka, što ga čini idealnim za moderne web aplikacije. Pored toga, ključni element poslužiteljske strane aplikacije je i Express.js - snažan i prilagodljiv Node.js okvir (engl. *framework*) koji nudi bogat skup značajki koje povećavaju produktivnost i olakšavaju razvoj web aplikacija - omogućava jednostavno organiziranje

funkcionalnosti aplikacije pomoću middlewarea i definiranja ruta, a pored toga i dodaje korisne alate Node HTTP objektima i olakšava izradu dinamičnih HTTP odgovora.

U narednom poglavlju rada opisane su tehnologije korištene za izradu AutoLoveRi aplikacije. Osim četiri temeljne tehnologije od kojih se sastoji zadani razvojni okvir, spomenute su i dodatne knjižnice čije je korištenje olakšalo razvoj i uljepšalo izgled same aplikacije. U trećem poglavlju opisan je rad same aplikacije, sa osvrtom na glavne funkcionalnosti koje aplikacija sadrži i koje su dostupne korisniku, te objašnjenjem razgraničenja između običnog korisnika i administratora. Četvrto poglavlje sadrži detaljan opis implementacije jedne od najzanimljivijih funkcionalnosti AutoLoveRi aplikacije, a to je dopisivanje u realnom vremenu između korisnika i administratora. U zadnjem poglavlju, odnosno zaključku, napisan je kratak zaključak rada u kojem se navode razlozi odabira tehnologija, kao i moguća buduća poboljšanja koja bi dodatno unaprijedila aplikaciju.

2 TEHNOLOGIJE

2.1 MongoDB

Pohrana i rad sa podacima AutoLoveRi aplikacije ostvareni su pomoću MongoDB [1] baze podataka. MongoDB kategoriziran je kao NoSQL (engl. *Not only SQL*) baza podataka kod koje se spremanje i dohvat podataka vrši u obliku kolekcija (engl. *collections*) i dokumenata (engl. *documents*). Baza podataka zasniva se na kolekcijama koje se sastoje od dokumenata. Pojedini dokumenti tvoreni su od zapisa (engl. *records*) koji sadrže informacije koje se žele pohraniti u obliku ključ-vrijednost (engl. *key-value*) parova. Umjesto pohranjivanja podataka u obliku tablica tj. njihovih redaka i stupaca kako se to čini u SQL bazama podataka, svaki zapis u MongoDB bazi podataka je dokument opisan u BSON-u (engl. *Binary JavaScript Object Notation*) čija binarna struktura kodira informacije o vrsti i duljini čime se omogućava mnogo brže pronalaženje u usporedbi sa JSON formatom.

Jedna od glavnih prednosti korištenja ove vrste baze podataka jest činjenica da su njeni dokumenti polimorfni. To znači da se polja pojedinih dokumenata unutar iste kolekcije mogu razlikovati, zbog čega se postiže viša razina fleksibilnosti na zahtjeve korisnika i uvelike pojednostavljuje proces modeliranja podataka. Osim toga, nema potrebe za predefiniranjem strukture dokumenta, već programeri mogu kodirati i spremati objekte u trenutku njihova stvaranja. Također, dodavanje novog polja dokumentu ne utječe na druge dokumente i ne uzrokuje isključivanje baze podataka što znači da se unesene promjene mogu neprimjetno pohraniti i ažurirati, bez potrebe za skupim operacijama kakve su ALTER TABLE ili redizajniranjem sheme od nule.

Za inkorporaciju MongoDB baze podataka u projektu korišten je MongoDB Atlas [2] – Database-as-a-Service (DbaaS) usluga baze podataka u oblaku koju pruža MongoDB, a koja omogućuje postavljanje, implementaciju i skaliranje baze podataka bez brige o lokalnom fizičkom hardveru, softverskim ažuriranjima i detaljima konfiguracije za performanse, na nekom od pružatelja usluga u oblaku (AWS, Azure ili GCP).

U konkretnom slučaju AutoLoveRi aplikacije, baza podataka sastoji se od ukupno 7 kolekcija: novi automobili, rabljeni automobili, korisnici, recenzije, narudžbe, razgovori i poruke.

2.1.1 Mongoose

Mongoose [3] je objektno-modelirajuća (engl. ODM – *Object Data Modeling*) knjižnica za Node.js koja omogućuje jednostavno korištenje MongoDB-a. Pomoću ove knjižnice, moguće je definirati sheme koje opisuju strukturu dokumenata unutar kolekcije, kao i izvršiti validaciju podataka. Jedna od bitnijih značajki ove knjižnice je i mogućnost referenciranja podataka između različitih kolekcija, čime se omogućava rad s povezanim podacima.

AutoLoveRi aplikacija zasnovana je na 7 Mongoose modela (podudarnih s ranije navedenim kolekcijama u bazi podataka), a pomoću značajke referenciranja izvršeno je povezivanje korisnika s njegovom narudžbom, administratora s mogućnošću dodavanja/brisanja vozila te razgovora i pojedinih poruka s korisnikom koji je njegov član, odnosno koji je poslao poruku.

2.1.2 BcriptJS knjižnica

BcryptJS [4] knjižnica predstavlja JavaScript implementaciju Bcrypt *hashing* algoritma [5] pomoću koje se postiže enkripcija korisničkih lozinki bez poznavanja kompleksnih hashing funkcija. *Hashing* predstavlja pretvorbu znakovnog niza u niz nasumičnih znakova čime se omogućava sigurno spremanje i prijenos osjetljivih podataka. Kao dodatni sloj zaštite u slučaju korištenja ove knjižnice, prije spremanja lozinke u bazu podataka, na ovako generiran znakovni niz dodaje se tzv. *salt* zbog kojeg je gotovo nemoguće dohvatiti prvobitnu lozinku. *Salt* predstavlja nasumični niz znakova koji je jedinstven za svaku lozinku i koji joj se dodaje prije samog *hashing* postupka, što rezultira time da iste lozinke imaju različite *hash* vrijednosti. Na ovaj način, uvelike je poboljšana zaštita lozinki koje se ponavljaju više puta u bazi podataka i otežano pogađanje lozinke na osnovu poznate *hash* vrijednosti.

Funkcionalnosti ove knjižnice u projektu implementirane su u sklopu definicije Mongoose modela za korisnika, koristeći dvije funkcije – za generiranje hashirane lozinke (*bcrypt.hash*) i provjeru podudaranja lozinke unesene od strane korisnika za vrijeme prijave i one spremljene u bazi podataka *(bcrypt.compare*).

2.2 Node.js

Node.js [6] predstavlja *cross-platform, runtime* okruženje za izvođenje JavaScript koda izvan web preglednika. Temelji se na V8 JavaScript engineu iz Google Chrome preglednika i predstavlja „JavaScript posvuda“ paradigmu, objedinjujući razvoj web-aplikacije oko istog programskog jezika i s poslužiteljske i s klijentske strane.

Jedna od glavnih prednosti korištenja ovog okruženja jest njegova skalabilnost koja mu omogućava rukovanje s velikim brojem istovremenih veza s malim resursima. Dok horizontalno skaliranje omogućuje pokretanje više instanci Node.js aplikacije, vertikalno skaliranje povećava snagu pojedinačne instance. Na taj način, dijeleći aplikaciju u više manjih instanci smanjuje se ukupno radno opterećenje i povećava personalizacija aplikacije jer je korisnicima moguće prikazati različite verzije aplikacija na temelju njihovih interesa, dobi, lokacije ili jezika. Pored toga, Node.js koristi neblokirajući, asinkroni I/O model pomoću kojeg je moguće rukovanje s velikim brojem istovremenih zahtjeva bez potrebe za čekanjem na završetak pojedinačnih operacija. Ova osobina Node.js-a od posebnog je značaja za aplikacije koje zahtijevaju brzu reakciju na korisničke zahtjeve ili trebaju obrađivati mnogo podataka u stvarnom vremenu.

Budući da se projekt AutoLoveRi temelji na MERN Stack razvojnom okviru, u njegovoj izradi korišten je Node.js radni okvir imena Express.

2.2.1 Express

Express [7] (ili Express.js) je najpopularniji radni okvir Node.js okruženja kojeg karakteriziraju jednostavnost korištenja i fleksibilnost. Iako je Node.js uveo JavaScript u programiranje na strani poslužitelja, nedostajale su mu ključne značajke poput usmjeravanja, izrade predložaka, međuprograma (engl. *middleware*) i rukovanja pogreškama. Budući da Express nudi sve od spomenutih značajki, relativno je brzo postao de facto standard za izgradnju poslužiteljskih aplikacija u Node.js ekosustavu.

Zahvaljujući usmjeravanju (engl. *routing*) omogućeno je jednostavno definiranje ruta za rukovanje s pristiglim HTTP zahtjevima, a pomoću implementacije ponovno iskoristivih *middleware* funkcija za presretanje zahtjeva i stvaranje odgovora olakšano je dodavanje funkcionalnosti poput autentifikacije i iščitavanja pristiglih podataka (engl. *data parsing*).

Express također nudi široku lepezu popratnih paketa za lakšu implementaciju dodatnih funkcionalnosti, od kojih u ovom projektu treba izdvojiti *cookie-parser* [8] za rad sa kolačićima (engl. *cookies*) i *cors* [9] (engl. Cross-Origin Resource Sharing) koji omogućava kontrolu nad time koji resursi mogu biti zatraženi s različitih domena, pružajući tako sigurniju komunikaciju.

2.3 React

Klijentski dio aplikacije izrađen je pomoću Reacta [10], JavaScript knjižnice za izgradnju korisničkih sučelja baziranih na komponentama. Koristeći React, aplikacije se razvijaju stvaranjem komponenti za višekratnu upotrebu koje se mogu vizualizirati kao neovisne „Lego kockice“. Korisničko sučelje je, dakle, kolekcija komponenti od kojih je svaka zadužena za ispisivanje malog dijela HTML koda. Velika prednost ovakvog pristupa izgradnji klijentskog dijela aplikacije je mogućnost ponovnog renderiranja samo onih dijelova stranice koji su se promijenili, što uvelike doprinosi poboljšanju korisničkog iskustva zbog veće brzine odaziva. Primarna uloga Reacta u aplikaciji je rukovanje slojem prikaza (engl. *view*), baš kao u obrascu *model-view-controller* (MVC), pružajući na taj način najbolje i najučinkovitije izvršenje renderiranja.

Za razliku od tipičnog modela razmjene podataka između web preglednika i poslužitelja u kojem se pri svakom novom zahtjevu za web stranicom ili njenim resursima dešava ponovno učitavanje cjelokupne stranice, React omogućava izradu tzv. jednostraničnih aplikacija (engl. single-page application, skraćeno SPA) koje na prvi zahtjev učitavaju jedan HTML dokument, a zatim po potrebi ažuriraju specifični dio zahvaćen promjenama.

React aplikacije se tipično sastoje od više slojeva komponenti koje se renderiraju u korijenski element DOM-a (engl. *Document Object Model*) korištenjem React DOM knjižnice. Prilikom renderiranja, zasebne komponente međusobno mogu prosljeđivati podatke koji se u Reactu nazivaju *props* (skraćeno od engl. *properties* – svojstva), dok se interne vrijednosti unutar komponente nazivaju njenim stanjem (engl. *state*). Pri tome, React se koristi virtualnim DOM-om koji predstavlja kopiju stvarnog i odmah se ponovno učitava kako bi odražavao svaku novonastalu promjenu u stanju podataka. Nakon toga, vrši se usporedba dvaju DOM-ova i pronalazi najučinkovitiji način implementiranja promjena u stvarni DOM.

Među bitne React značajke spadaju i *React Hooks* [11] – posebne funkcije kojima se postiže manipulacija stanja unutar komponente, ali i dijeljenje ponašanja između komponenti

bez potrebe za nasljeđivanjem. Neke od najznačajnijih su *useState* (upravlja lokalnim stanjem komponente), *useEffect* (upravlja ponašanjem komponente pri realizaciji nekog vanjskog efekta) i *useContext* (omogućava korištenje React konteksta unutar komponenti).

2.4 Tailwind CSS

Stiliziranje funkcionalnih React komponenti s ciljem vizualno privlačnijeg korisničkog sučelja implementirano je pomoću CSS (engl. *Cascading Stye Sheets*) radnog okvira imena Tailwind CSS [12]. Tailwind CSS radi na principu skeniranja JavaScript komponenti za nazive klasa pomoću kojih generira odgovarajuće stilove i zatim ih zapisuje u statičnu CSS datoteku. Zasnovana je na tzv. *utility-first* principu, što znači da se ne oslanja na unaprijed definirane klase kakve se koriste u samom CSS-u, nego pruža bogat set *utility* klasa koje se koriste za stiliziranje elemenata. Osnovne postavke dizajna koji se želi odraziti u cijeloj aplikaciji, kao što je boja pozadine, veličina teksta ili font mogu se postaviti putem konfiguracijske datoteke (*tailwind.config.js*) čime se postiže koherentnost i dosljednost izgleda web stranice. Osim ljepšeg estetskog dojma, *utility* klase TailwindCSS knjižnice pružaju i mogućnost responzivnosti statičkog dizajna, odnosno mijenjanje njegova izgleda prilikom interakcije ili promjene veličine ekrana.

2.5 Jsonwebtoken

Jsonwebtoken [13] predstavlja implementaciju JSON Web Tokena – kompaktnih tokena koji koriste JSON, a služe za prijenos podataka na siguran i verificiran način. Ova knjižnica koristi se u svrhe autentifikacije i autorizacije, a svaki od tokena sastoji se od 3 dijela: zaglavlja, tijela i potpisa. Zaglavlje tokena sadrži informaciju o algoritmu koji se koristi za potpisivanje te tipu tokena. Tijelo tokena sadrži podatke koji se prenose, a potpis se koristi za provjeru autentičnosti tokena.

U konkretnom primjeru AutoLoveRi web stranice, tokeni su generirani prilikom registracije i prijave korisnika te pohranjeni u kolačiće web preglednika te korišteni za odvajanje korisničkog i administratorskog dijela aplikacije, a brisani iz kolačića prilikom odjave korisnika.

2.6 Socket.IO

Funkcionalnost dopisivanja korisnika s administratorom u realnom vremenu ostvarena je pomoću Socket.IO JavaScript knjižnice [14]. Dizajnirana kako bi pojednostavila rad s WebSocketima [15] i osigurala brzu i pouzdanu komunikaciju u stvarnom vremenu, ova knjižnica posebno je korisna u aplikacijama koje zahtijevaju stalnu sinkronizaciju podataka.

U tradicionalnom HTTP-u, klijent šalje zahtjev poslužitelju koji mu odgovara traženim podacima. Ovakav model komunikacije zahtijeva kontinuirano prozivanje (engl. *polling*) od klijenta prema poslužitelju, što može rezultirati povećanom latencijom i smanjenom učinkovitošću. Suprotno tome, WebSockets uspostavljaju trajnu vezu između klijenta i poslužitelja, što znači da nakon što se veza jednom uspostavi, obje strane mogu slati podatke jedan drugome u bilo kojem trenutku bez potrebe za stalnim prozivanjem.

U slučaju da se veza putem WebSocketa ne može ostvariti, Socket.IO knjižnica vratit će se HTTP prozivanju kako bi i u tom slučaju nastavila sa prijenosom podataka. Ukoliko se pod specifičnim uvjetima prekine veza između klijenta i poslužitelja bez da su oni toga svjesni, Socket.IO implementira mehanizam za periodičnu provjeru stanja veze i automatsko ponovno povezivanje sa eksponencijalnim zakašnjenjem, čime se izbjegava preopterećenost poslužitelja.

2.7 Ostale korištene knjižnice i paketi

2.7.1 AOS

AOS (engl. *animate on scroll*) [16] knjižnica omogućava jednostavno dodavanje animacija vidljivih prilikom pomicanja prema dolje, odnosno scrollanja po web stranici. AOS knjižnica kao takva pruža jednostavan i učinkovit način za stvaranje dinamičnog i privlačnog web dizajna, bez potrebe za opsežnim poznavanjem složenih tehnika animacije.

2.7.2 React-icons i React-Toastify

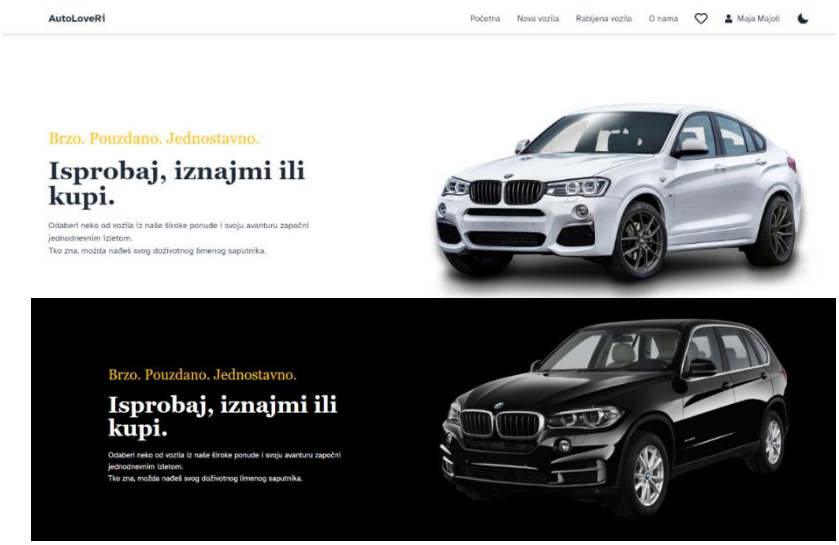
Za dodatno poboljšanje korisničkog iskustva prilikom korištenja aplikacije, korištene su popularne knjižnice – *react-icons* [17] i *react-toastify* [18]. Knjižnica *react-icons* omogućava jednostavno korištenje ikona iz popularnih ikonografskih setova u React aplikacijama, a zahvaljujući činjenici da omogućava uvoz samo onih ikona koje su potrebne, smanjuje se veličina konačne aplikacije. Također, sve ikone se koriste kao React komponente, zbog čega se mogu lako umetnuti u JSX kod ali i stilizirati.

Knjižnica *react-toastify* pomaže sa upravljanjem obavijestima (engl. *toasts*) u React aplikacijama u obliku skočnih prozora koji se odgovarajućom stilizacijom mogu dodati na željeno mjesto na ekranu. U slučaju AutoLoveRi projekta, ovakve se obavijesti koriste kako bi npr. korisnik imao povratnu informaciju u slučaju unosa neispravnih vrijednosti za vrijeme prijave ili registracije, ali i u mnogim drugim slučajevima.

3 AutoLoveRi - APLIKACIJA ZA PRODAJU NOVIH I RABLJENIH AUTOMOBILA

3.1 Početna stranica aplikacije

Početna stranica AutoLoveRi aplikacije za prodaju novih i rabljenih automobila sadrži osnovne informacije o tvrtki i njenom načinu poslovanja. Prvo što korisnik vidi pri otvaranju ove stranice je naslovni element s nazivom tvrtke i kratkim slogan sa jedne, te slikom jednog od automobila s druge strane. Na samom vrhu stranice nalazi se navigacijska traka pomoću koje se može navigirati do stranice s dodatnim informacijama o ovoj kompaniji ili do neke od stranica s ponudom trenutno dostupnih vozila. Također, klikom na ikonu u obliku sunca (za svijetli način prikaza) ili mjeseca (za tamni način prikaza), korisnik može shodno svojim preferencijama promijeniti temu. Ova funkcionalnost implementirana je pomoću varijable *theme* pohranjene u lokalnom međuspremniku (engl. *local storage*) definirane na razini korijenske komponente *App*. Prosljeđujući ovu varijablu kao argument *context* svojstva *Outlet* komponente, omogućen je odraz njene promjene kroz cijelu aplikaciju. Izgled navedenih elemenata vidljiv je na slici 3.1.

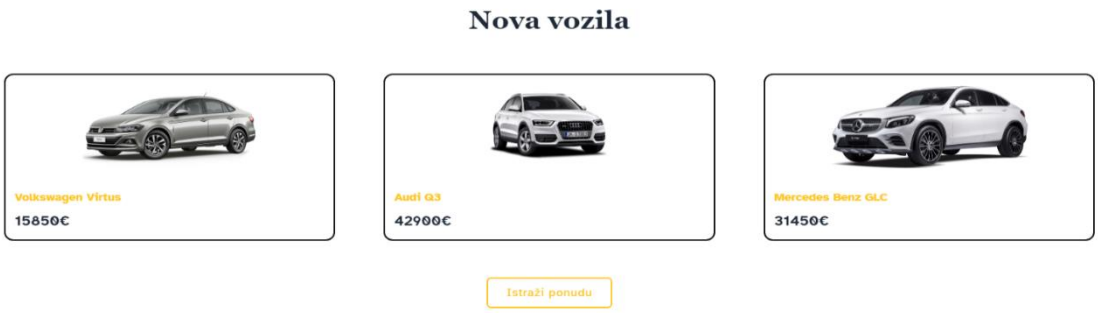


Slika 3.1. Naslovni element u svijetlom (gore) i tamnom (dolje) načinu rada

Kondicionalno renderiranje se također koristi i za prikaz imena trenutno registriranog korisnika ili teksta 'Prijavi se', ukoliko nitko nije trenutno prijavljen. Također, omogućeno je dinamičko prikazivanje broja stavki u korisničkoj listi želja koje se ažurira svaki put kada korisnik odabere dodati novo vozilo.

Na početnoj stranici nalazi se i komponenta za odabir rubrike novih ili rabljenih vozila, koja također provodi usmjeravanje na za to predviđene stranice, a ispod nje nalazi se i komponenta koja ukratko opisuje prednosti kupovine automobila putem AutoLoveRi stranice.

Najznačajniji element ove stranice je komponenta s prikazom novih vozila, koja prikazuje tri najnovije dodana automobila u toj kategoriji. Svaki od automobila prikazan je u zasebnom elementu (kartici), u kojoj se vidi slika vozila, naziv marke i modela, te njegova cijena. Korištenjem istog elementa za svaki od automobila, do izražaja je došla jedna od glavnih prednosti korištenja React knjižnice – mogućnost upotrebe jednom definirane komponente više puta, bez potrebe višestrukog ponavljanja koda, čime se postiže kompaktniji i pregledniji kod. Ispod automobila nalazi se gumb čijim pritiskom korisnik može vidjeti i sva ostala vozila iz ponude. Izgled ove komponente vidljiv je na slici 3.2.

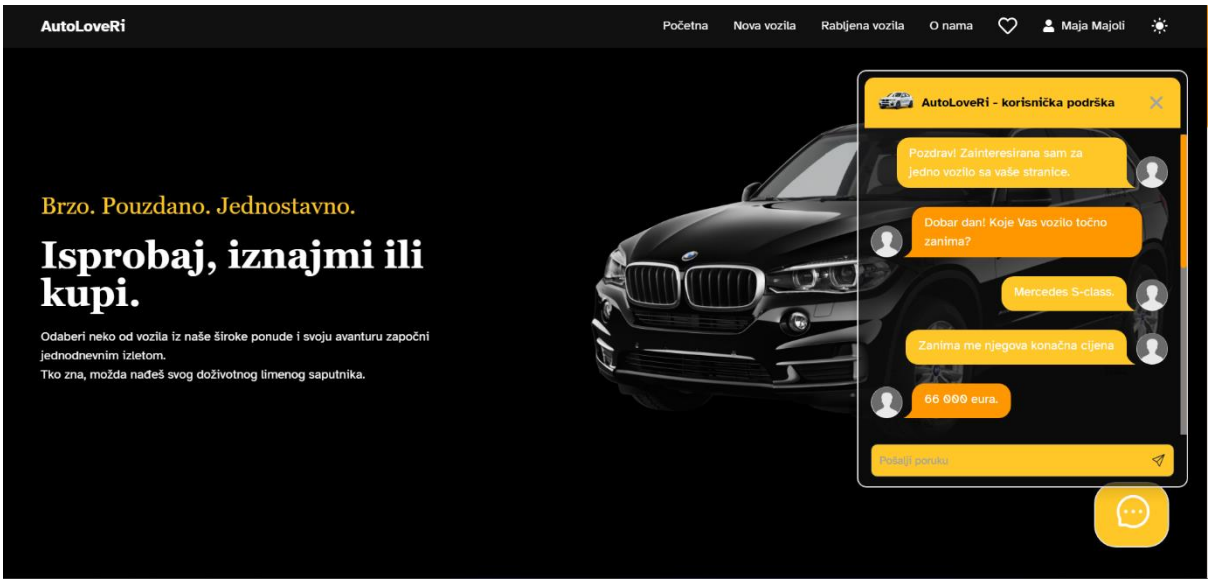


Slika 3.2. Prikaz tri najnovije dodana nova automobila

Na samom dnu početne stranice locirana je sekcija u kojoj su vidljive recenzije zadovoljnih korisnika, čiji se sadržaj također dohvaća iz baze podataka, a nakon nje vidljivo je i podnožje (engl. *footer*) aplikacije, koje zajedno sa navigacijskom trakom, dijele sve stranice. Komponenta podnožja sastoji se od odjeljka sa osnovnim kontakt informacijama AutoLoveRi tvrtke, a preostali stupci vrše usmjeravanje korisnika na ranije spomenutu stranicu s ponudom novih ili rabljenih vozila.

Osim navedenih komponenti, u donjem desnom uglu stranice nalazi se komponenta putem koje se realizira dopisivanje u realnom vremenu iz perspektive korisnika. Klikom na ikonu otvara se skočni prozor u kojem su vidljive ranije razmijenjene poruke s administratorom (ukoliko one postoje) ili prazan prozor ukoliko korisnik po prvi put započinje razgovor sa korisničkom podrškom. Ovim putem, korisnik se može dodatno informirati o vozilu za koje je zainteresiran ili stanju svoje narudžbe. Estetska strana ove

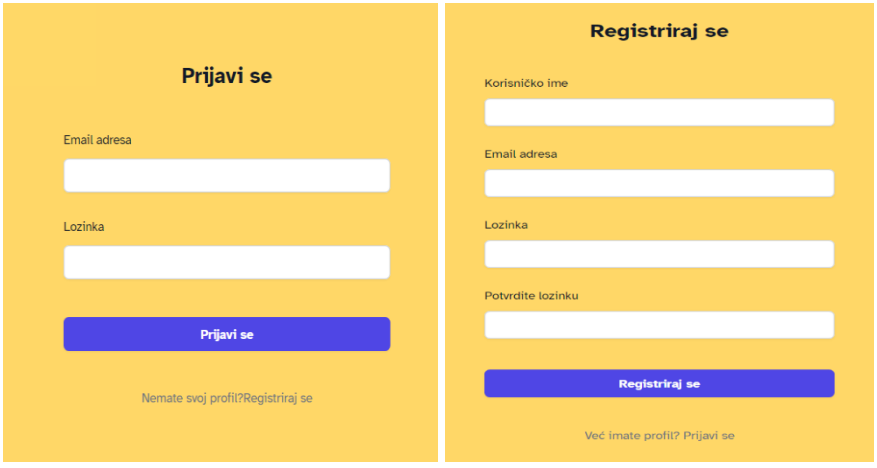
funkcionalnosti vidljiva je na slici 3.3, a o njenoj implementaciji bit će govora u kasnijem poglavlju.



Slika 3.3. Dopisivanje s administratorom u vidu skočnog prozora

3.2 Prijava i registracija korisnika

Registracija korisnika vrši se putem jednostavnog *form* elementa u kojem se od njega traži da unese korisničko ime, svoju email adresu i lozinku koju naposlijetku mora još jednom potvrditi. Kako bi korisnik imao što bolji uvid u eventualne pogreške nastale prilikom registracije, povratne informacije prikazane su mu u obliku skočnog prozora u gornjem desnom uglu. Putem njega, korisnik može doznati ukoliko već postoji račun sa unesenom email adresom, ukoliko su uneseni podaci neispravni (email adresa bez @ znaka) ili se lozinke ne podudaraju. Ispod gumba za registraciju korisnika nalazi se i poveznica koja korisnika sa već kreiranim profilom odvodi na ekran za prijavu. Izgled forme putem koje se vrši prijava i registracija vidljiv je na slici 3.4.



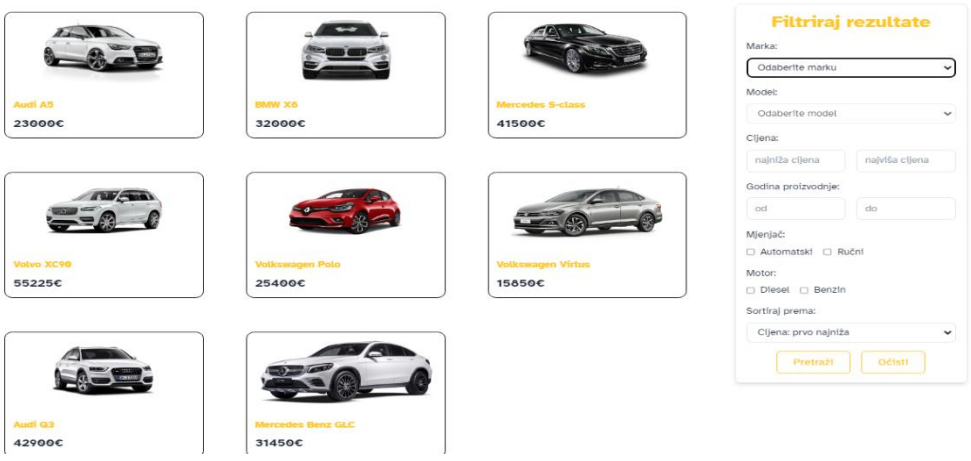
Slika 3.4. Izgled forme za prijavu (lijevo) i registraciju (desno)

Prijava korisnika vrši se na osnovu unesene email adrese i lozinke. U slučaju da korisnik unese neispravne podatke, pojavljuje se poruka 'Neispravan email ili lozinka', čime se smanjuje mogućnost hakiranja profila, budući da se ne zna koja od dvije stavke je netočna. Za autentifikaciju i autorizaciju registriranih i prijavljenih korisnika u AutoLoveRi aplikaciji korišten je JSON Web Token, o čijem načinu rada će biti govora u kasnijem poglavlju.

3.3 Ponuda novih i rabljenih vozila

Na stranici koja prikazuje ponudu novih vozila korisnik ima mogućnost vidjeti sva trenutno dostupna vozila iz ove kategorije. Osim toga, s desne strane zaslona nalazi se komponenta pomoću koje je omogućeno filtriranje i sortiranje prikazanih podataka. Ova se komponenta na manjim zaslonima pojavljuje iznad one s ponudom automobila, čime se postiže bolji korisnički doživljaj ukoliko se aplikaciji pristupa npr. putem mobilnog uređaja. Putem ove komponente korisnik može odabrati kriterije koje vozilo mora proći da bi bilo prikazano, čime se pronalazak njegova idealnog vozila čini još jednostavnijim.

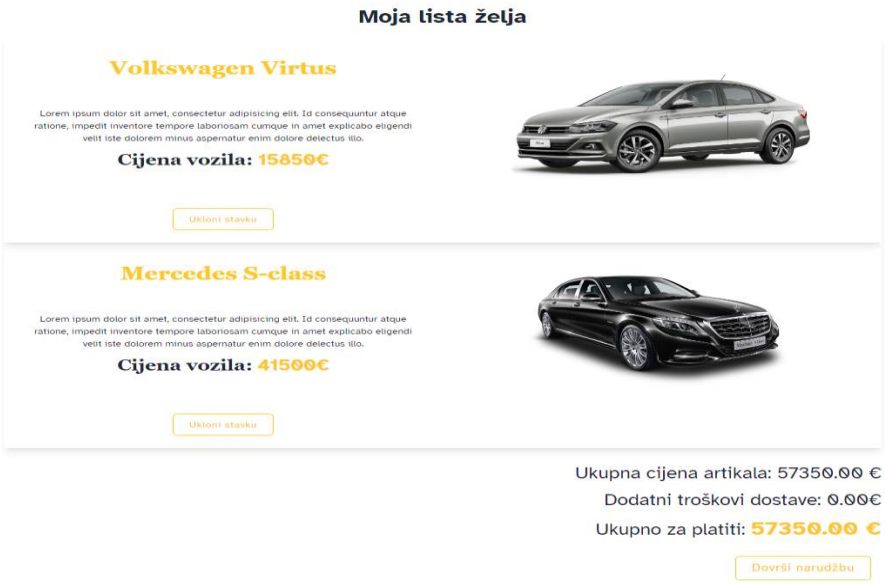
Prikazane podatke moguće je sortirati prema cijeni u uzlaznom i silaznom redoslijedu. Dodatno, ali i neovisno od toga, podaci se mogu filtrirati po proizvođaču automobila, modelu, rangu cijene i godine proizvodnje te mjenjaču i motoru. Ponuđene marke automobila sastoje se samo od onih trenutno postojećih u bazi podataka, a nakon odabira marke u ponudi modela prikazuju se samo modeli te odabrane marke. Ukoliko korisnik definira kriterije filtriranja kojima ne odgovara niti jedno vozilo iz baze podataka, umjesto podataka prikazati će mu se odgovarajuća poruka – „Nažalost, niti jedno novo vozilo u našoj ponudi ne odgovara Vašim kriterijima“. Na dnu komponente nalaze se gumbi pomoću kojih se istovremeno mogu pobrisati svi filteri kako bi se krenulo ispočetka i za prikaz željenih vozila. Izgled stranice za ponudu novih automobila vidljiv je na slici 3.5.



Slika 3.5. Stranica sa ponudom novih vozila

3.4 Proces naručivanja vozila i naplata narudžbe

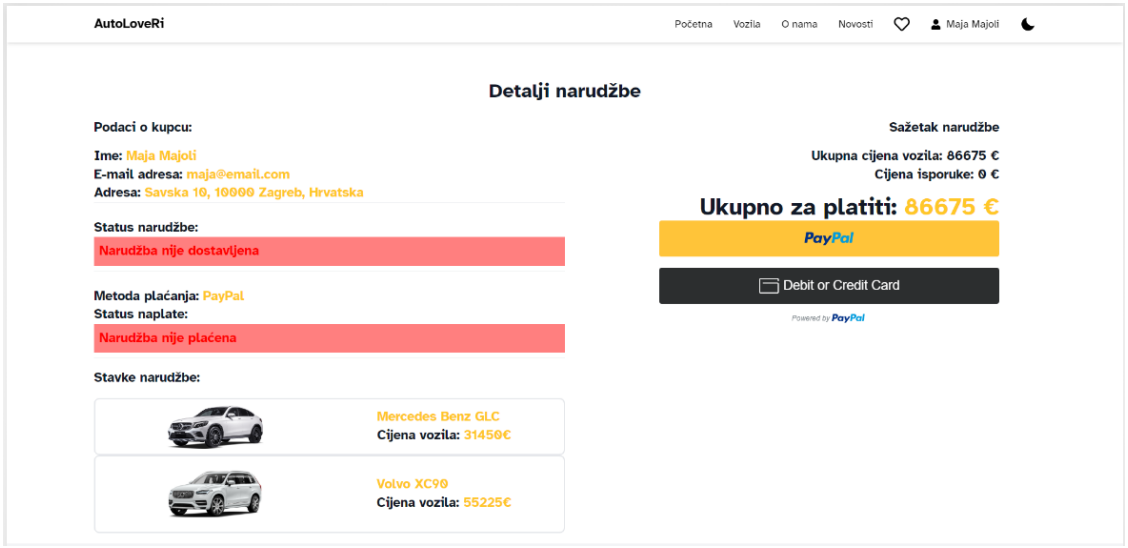
Kada korisnik odabere vozilo koje želi naručiti, pritiskom na gumb *Naruči odmah* željeno vozilo sprema u svoju košaricu (listu želja). Klikom na ikonu u obliku srca koja se nalazi u navigacijskoj traci, a čiji se popratni broj mijenja sukladno broju vozila koje je korisnik odabrao, dolazi se na ekran koji prikazuje trenutnu listu želja, vildljiv na slici 3.6.



Slika 3.6. Izgled korisničke liste želja sa odabranim vozilima

Važno je napomenuti da na taj ekran može doći samo prijavljeni korisnik – ukoliko mu pokuša pristupiti bez prijave, izvršit će se direktna redirekcija na ekran za prijavu. Ukoliko se predomisli, klikom na gumb *Ukloniti stavku* korisnik može maknuti vozilo iz popisa želja. Pri dnu stranice nalazi se ukupna suma koju korisnik treba platiti za svoju narudžbu – ukoliko je narudžba skuplja od 50 tisuća eura, korisnik mora platiti dodatnih tisuću za isporuku. Popis artikala u košarici/listi želja kao i njihova cijena, nalaze se pohranjeni u *cart* stanju *local storagea,* zajedno s adresom na koju korisnik želi izvršiti isporuku vozila i načinom plaćanja. Za izračun cijena narudžbe i isporuke koristi se logika izdvojena u zasebnu datoteku naziva cartUtils.js – funkcija *updateCart*() sadrži ukupno tri koraka – izračun cijene vozila koji se računa kao suma *cartItems* artikala, zatim izračun cijene isporuke po gore navedenoj logici i izračun ukupne cijene kao sume te dvije prethodne. *UpdateCart*() funkcija poziva se svaki put kada se desi neka promjena u košarici – bilo da je to dodavanje artikla, njegovo uklanjanje ili unos adrese isporuke i plaćanje narudžbe.

Cjelokupan proces naplate i slanja narudžbe odvija se u 4 koraka – s ekrana košarice pritiskom na gumb *Dovrši narudžbu* korisnik dolazi na ekran isporuke u kojem se od njega traži unos ulice, poštanskog broja, grada i države, a sve se te informacije spremaju u gore spomenuti *cart*. Ukoliko ovo nije prvi put da korisnik nešto naručuje, forma će se automatski ispuniti njegovim prethodnim podacima, ali ih on i dalje po potrebi može promijeniti. Potom slijedi ekran za odabir načina plaćanja, koji je u ovom slučaju PayPal budući da je izvršena njegova integracija u aplikaciju. Naposlijetku korisnik još jednom može vidjeti sažetak svoje narudžbe (naručene artikle, cijenu, svoje podatke i na kraju ukupnu cijenu) te klikom na *Završi narudžbu* doći do ekrana s detaljima narudžbe koji je vidljiv na dolje priloženoj slici 3.7. Početno stanje svake narudžbe je da nije plaćena niti dostavljena, kako je naznačeno crveno obojanim elementima.



Slika 3.7. Prikaz ekrana sa detaljima narudžbe

Po uspješnoj naplati od strane korisnika, status naplate preći će u *Narudžba je plaćena*, označen zelenom bojom. S druge strane, promjena statusa narudžbe može biti promijenjena samo od strane administratora, i to samo nakon što je izvršena uplata.

3.5 Administratorske ovlasti

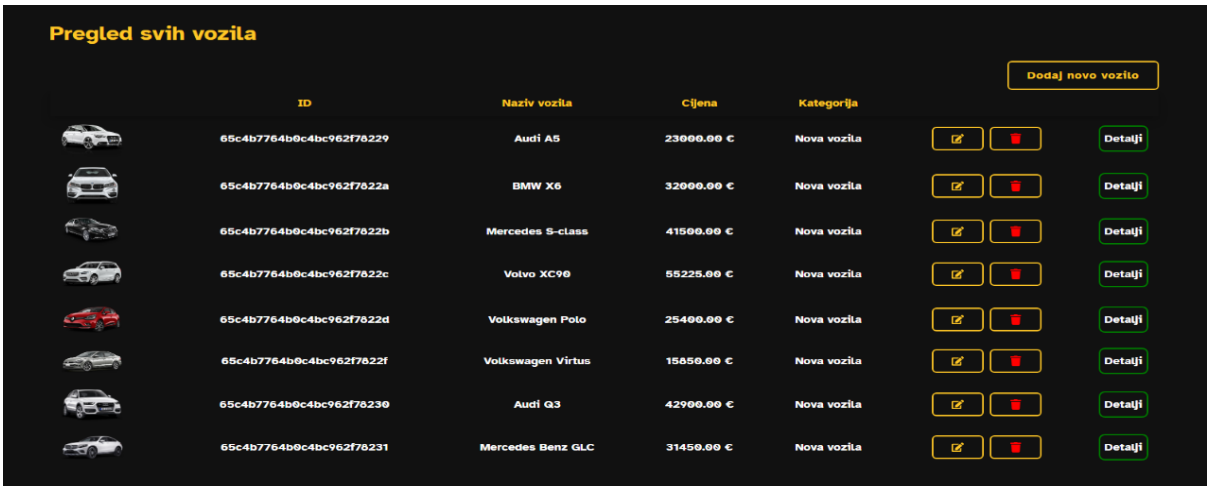
3.5.1 Upravljanje korisnicima, vozilima i narudžbama

Korisnik definiran kao administrator ima dodatne ovlasti koje regularni korisnik nema. Klikom na svoje ime u desnom gornjem uglu navigacijske trake, otvara se padajući meni koji se sastoji od ukupno 5 opcija. Dvije od tih opcija imaju svi korisnici, a to su: Moj profil – poveznica na ekran pomoću kojeg svi korisnici mogu ažurirati svoje podatke, odnosno

promijeniti svoje ime ili lozinku, i Odjava – poveznica kojom se izvršava ranije spomenuta *logout* akcija i dolazi do brisanja korisničkih podataka iz lokalnog spremnika. Tri opcije koje ga razlikuju od ostalih korisnika jesu poveznice Narudžbe, Vozila i Korisnici. Klikom na opciju Korisnici, administrator može vidjeti popis svih trenutno registriranih korisnika iz baze podataka, te ih po potrebi može obrisati ili im dodijeliti ovlasti administratora. Kako to izgleda može se vidjeti na slici 3.8.

Slika 3.8. Pregled svih korisnika

Pod opcijom Vozila nalaze se dva popisa – jedan za rabljena i jedan za nova vozila. Admin može vidjeti identifikacijski broj vozila, njegovo ime i cijenu, a ima i mogućnosti brisanja vozila, ažuriranja njegovih podataka kao i direktnu poveznicu na stranicu sa detaljima tog vozila. Osim toga, on može i dodati novo vozilo. Dodavanje novog vozila vrši se na način da se u bazu podataka unese vozilo sa određenim početnim (tzv. *dummy*) vrijednostima, koje admin potom može pretvoriti u željene klikom na gumb *Uredi*. Na dolje umetnutoj slici 3.9. prikazan je ekran sa popisom novih vozila. Jedna od bitnijih stavki ove administratorske ovlasti jeste mogućnost ažuriranja podataka vozila u sklopu koje se može ostvariti i ažuriranje same slike vozila, a to je ostvareno koristeći *multer* [21], Node.js middleware za učitavanje datoteka.



Slika 3.9. Pregled svih vozila

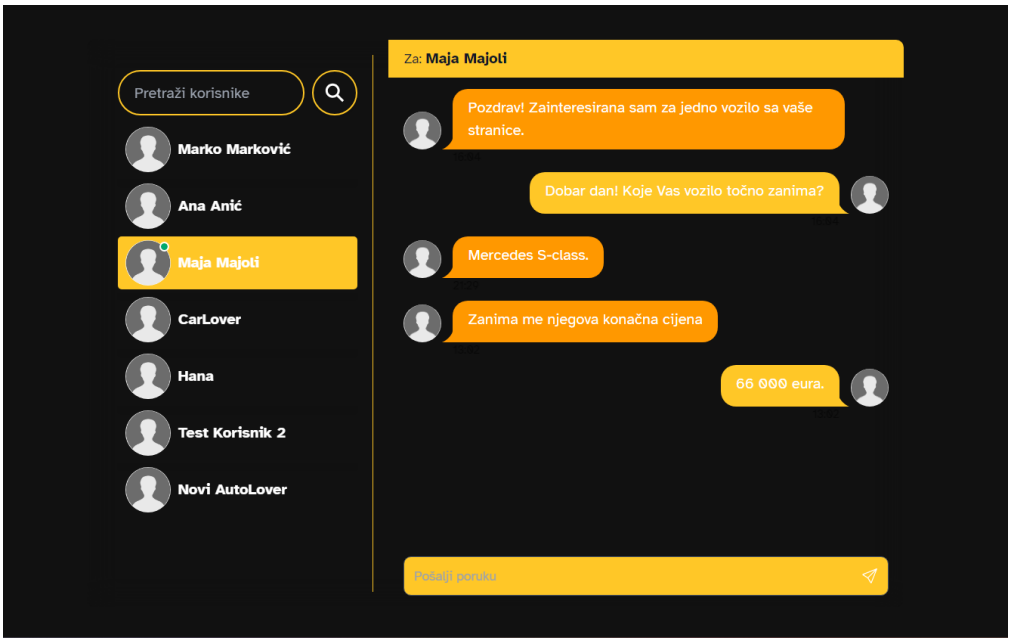
Treća ovlast dostupna samo administratoru je prikaz svih dosadašnjih narudžbi. Kao što se može vidjeti na slici 3.10 ispod teksta, i ona se zasniva na prikazu identifikacijskog broja narudžbe, imena korisnika koji je izvršio narudžbu, datum narudžbe i njena cijena. Osim toga, prikazani su status naplate i isporuke, koji su u početnom stanju obilježeni crvenim križićima. Tek nakon što korisnik izvrši uplatu narudžbe, putem PayPala integriranog u aplikaciji, crveni križić biva zamijenjen datumom uplate iznosa i administrator ima mogućnost ažurirati status narudžbe na *Isporučeno*, tako što klikom na gumb *Detalji* ode na ekran sa detaljima narudžbe, otkud može promijeniti status isporuke.

Slika 3.10. Pregled svih narudžbi

3.5.2 Dopisivanje s korisnikom u realnom vremenu

Ostvarivanje funkcionalnosti dopisivanja sa administratorom aplikacije, u vidu korisničke podrške putem koje korisnik može dobiti dodatne informacije o vozilu ili svojoj narudžbi, implementirano je korištenjem React komponenti koje su stilizirane pomoću daisyUI [22] knjižnice.

Iz administratorske perspektive ova je funkcionalnost dodana u vidu komponente koja se prostire preko cijelog ekrana i ima izgled tipične aplikacije za dopisivanje, sa popisom korisnika sa lijeve i dijelom za razmjenu poruka sa desne strane, a njen je izgled vidljiv na slici 3.11. Osim popisa razgovora sa korisnicima, sa lijeve strane nalazi se i komponenta za pretraživanje korisnika, koja omogućava administratoru lakši pronalazak specifičnog korisnika. Funkcija za pretraživanje razgovora zahtijeva od administratora da unese najmanje 3 znaka. U slučaju da taj uvjet nije ispunjen, ili uneseni pojam ne odgovara imenu niti jednog od postojećih korisnika, u gornjem uglu ekrana pojavljuje se odgovarajuća poruka.



Slika 3.11. Dopisivanje s korisnikom (administrator)

Također, korisnici koji su trenutno aktivni (odnosno u datom trenutku posjećuju AutoLoveRi stranicu), obilježeni su zelenim kružićem koji ukazuje na status njihove aktivnosti.

4 RAZVOJ I IMPLEMENTACIJA SPECIFIČKIH APLIKACIJSKIH FUNKCIONALNOSTI

4.1 Autentifikacija korisnika putem JSON Web Tokena

Izvedba ove funkcionalnosti AutoLoveRi aplikacije izvršena je putem JWT (engl. *JSON Web Token*) u koji se sprema ID korisnika. Taj token sprema se u obliku HTTP-only kolačića na serveru, umjesto u lokalnom međuspremniku, za manju izloženost mogućim curenjima podataka. Trajanje tokena je pritom postavljeno na 30 dana. Generiranje tokena prilikom procesa prijave i registracije realizirano je putem funkcije *generateTokenAndSetCookie()* čiji je kod vidljiv na slici 4.1.

```
const generateTokenAndSetCookie = (userId, res) => {
    //userId = payload (digital signature - how a token get assigned
    const token = jwt.sign({userId}, process.env.JWT_SECRET, {
        expiresIn: '30d',
    })

    //set this token as a cookie named "jwt"
    res.cookie("jwt", token, {
        maxAge: 30 * 24 * 60 * 60 * 1000,
        httpOnly: true,
        sameSite: "strict",
        secure: process.env.NODE_ENV !== "development"
    })
}
```

Slika 4.1. Programski kod funkcije odgovorne za generiranje autentifikacijskog tokena

Kako bi se jasno razgraničio dio aplikacije dostupan korisniku i onaj namijenjen samo administratoru, kreirane su dodatne 2 funkcije koje koriste tokene - jedna za stvaranje zaštićenih ruta (onih kojima mogu pristupiti samo registrirani korisnici) – naziva *protect*, i jedna za detekciju administratora (jer jedino on može npr. vidjeti sve registrirane korisnike) – naziva *admin*. Zaštićene su one rute koje vode do profila trenutnog korisnika ili do mogućnosti da se isti taj profil ažurira od strane njega samoga. U slučaju da korisnik želi pristupiti ruti koja je zaštićena admin *middleware*-om, dobit će poruku „*Niste ovlašteni, neispravan token*“ sa statusom 401 (*Unauthorized*), kao i za slučaj da njihov token uopće ne postoji.

Odjavljivanje (engl. *logout*) korisnika u ovom slučaju znači brisanje JWT kolačića, a ta se funkcionalnost u aplikaciji realizira tako da se u tijelo kolačića pošalje prazan znakovni niz, u *options* dio stavka *expires* postavi u vrijednost *new Date(0)* i naposlijetku pošalje odgovarajuća poruka uz status 200 (Uspješno odjavljivanje). Također, po završetku

registracije novog korisnika, automatski se izvrši i njegovo prijavljivanje, odnosno postavlja se njegov kolačić. Programski kod pomoću kojeg se implementira funkcionalnost odjavljivanja korisnika i brisanja njegovog kolačića vidljiv je na slici 4.2.

```
// @desc logs out the user & deletes the cookie
// @route POST /api/users/logout
// @acces private
const logoutUser = asyncHandler (async (req, res) => {
    res.cookie('jwt', '', {
        httpOnly: true,
        expires: new Date(0),
    });

    res.status(200).json({ message: 'Uspješna odjava' });
});
```

Slika 4.2. Programski kod za odjavu korisnika

4.2 Plaćanje putem PayPala

4.2.1 Integracija PayPala (poslužiteljska strana)

Za integraciju, odnosno simulaciju plaćanja narudžbe koristi se PayPal. Putem PayPal Developer [19] stranice stvoren je poslovni račun na koji će pristizati novac naplaćenih narudžbi. U *.env* datoteku iz tog je razloga uveden *PAYPAL_CLIENT_ID*, koji je preuzet s iste te stranice. Najprije se integracija PayPal funkcionalnosti odvila s poslužiteljske strane, kreiranjem rute na poslužitelju pomoću koje PayPal dolazi do navedenog ID-a, jer se kao odgovor na zahtjev poslan na tu rutu vraća JSON objekt koji sadrži *clientId* polje čija je vrijednost upravo ranije navedeni ID broj. Programski kod za uspostavljanje ovog endpointa vidljiv je na slici 4.3. Kasnije je taj *endpoint* korišten od klijentske strane aplikacije, kako bi se inicijaliziralo PayPal plaćanje odgovarajućeg korisnika.

```
//paypal
app.get('/api/config/paypal', (req, res) => res.send({ clientId: procces.env.PAYPAL_CLIENT_ID}));
```

Slika 4.3. Definiranje rute za dohvat PayPal ID-a

Sami proces plaćanja narudžbe sa poslužiteljske strane poziva se korištenjem payOrder mutacije, odnosno izvršavanjem updateOrderToPaid() funkcije pomoću koje se mijenja status narudžbe u plaćeno i vrijeme naplate stavlja na ono trenutno. Kod spomenute updateOrderToPaid() funkcije vidljiv je na slici 4.4.

```
const updateOrderToPaid = asyncHandler (async (req, res) => {
    const order = await Order.findById(req.params.id);

    if (order) {
        order.isPaid = true;
        order.paidAt = Date.now();
        order.paymentResult = {
            _id: req.body.id,
            status: req.body.status,
            update_time: req.body.update_time,
            email_address: req.body.payer.email_address,
        };

        const updatedOrder = await order.save();
        res.status(200).json(updatedOrder);

        } else {
            res.status(404);
            throw new Error ('Order not found')
    }
});
```

Slika 4.4. Poslužiteljska strana naplate narudžbe

4.2.2 Integracija PayPala (klijentska strana)

Za implementaciju na klijentskoj strani korišten je paket *@paypal/react-paypal-js* [20], za upotrebu kojeg je potrebno cijelu aplikaciju "umotati" (engl. *wrap*) u element *PayPalScriptProvider*. Postavljanjem svojstva *deferLoading* ovog elementa na vrijednost *false*, omogućeno je automatsko učitavanje PayPal SDK skripte prilikom renderiranja. PayPal SDK (engl. *Software Development Kit*) je zbirka softverskih alata koja programerima omogućava integraciju PayPalovih usluga u njihove aplikacije i web stranice te olakšava proces dodavanja opcija za plaćanje i upravljanje transakcijama putem PayPala. Budući da se AutoLoveRi aplikacija temelji na JavaScriptu, za implementaciju plaćanja korišten je PayPal JavaScript SDK. Osim pokretanja odgovarajuće SDK skripte, *@paypal/react-paypal-js* omogućava i korištenje komponenti odgovornih za prikaz PayPal gumba na klijentskoj strani aplikacije.

Plaćanje putem PayPala odvija se na posljednjem od 4 koraka izvršavanja narudžbe, onom koji sadrži njen sažetak (adresu isporuke, odabrano vozilo/a i konačnu cijenu) i navedene gumbe. Prilikom klika na gumbe najprije se izvrša *createOrder()* funkcija odgovorna za kreiranje novog PayPal računa za plaćanje. Programski kod ove funkcije prikazan je na slici 4.5, a iz njega se vidi da se otvara račun iznosa *order.totalPrice* (pri čemu je order jedan od modela iz baze podataka koji sadrži sve podatke pojedine narudžbe), a vraća se ID broj kreirane narudžbe.

```
function createOrder(data, actions) {
    return actions.order.create({
        purchase_units: [
            {
                amount: {
                value: order.totalPrice
                },
            },
        ],
    }).then((orderId) => {
        return orderId
    })
}
```

Slika 4.5. Programski kod za kreiranje PayPal računa

Kada korisnik uspješno odobri plaćanje, izvršava se funkcija *onApprove()* pomoću koje se potvrđuje i završava transakcija (*actions.order.capture()*). Nakon toga, poziva se funkcija *payOrder* implementirana na poslužiteljskoj strani. Za bolje korisničko iskustvo, povratna informacija o (ne)uspjehu izvršene transakcije, prikazuje se kupcu u obliku skočnog prozora kreiranog pomoću *react-toastify* knjižnice. Implementacija ove funkcionalnosti vidljiva je na slici 4.6.

```
function onApprove(data, actions) {
    return actions.order.capture().then(async function (details) {
        try {
            await payOrder({orderId, details});
            refetch();
            toast.success('Uspješno plaćeno')
        } catch (err) {
            toast.error(err?.data?.message || err.message)
        }
    });
}
```

Slika 4.6. Programski kod za naplatu transakcije

4.3 Dopisivanje u realnom vremenu

4.3.1 Modeli podataka: poruka i razgovor

Podaci spremljeni u bazi podataka koji se koriste za realizaciju dopisivanja baziraju se na dva Mongoose modela – poruka (engl. Message) i razgovor (engl. Conversation). Model poruke sastoji se od identifikacijskog broja pošiljatelja i primatelja te tijela poruke koja se želi prenijeti. Pri tome, koristeći se referenciranjem, identifikacijski brojevi povezani su sa modelom korisnika (engl. User) koji se koristi za definiciju svih registriranih korisnika. Ovaj

model koristi se i vremenskim oznakama koje se u daljnjoj implementaciji koriste za prikazivanje točnog vremena kad je poruka poslana.

Razgovor između dva korisnika sprema se na osnovu modela razgovora koji u sebi sadrži 2 polja objekata: jedno imena učesnici (engl. participants) u kojem su pohranjeni identifikacijski brojevi oba korisnika koji sudjeluju u razgovoru (također referenciranjem povezani sa modelom korisnika) i drugo imena poruke (engl. messages) koje se sastoji od jedinstvenih oznaka poruka koje su dio ovog razgovora (ponovno, referenciranjem se ovaj model povezuje sa modelom poruke).

4.3.2 Inicijalizacija Socket.IO poslužitelja

Ostvarivanje funkcionalnosti dopisivanja sa administratorom aplikacije, u vidu korisničke podrške putem koje korisnik može dobiti dodatne informacije o vozilu ili svojoj narudžbi, implementirano je korištenjem Socket.IO knjižnice. Integracija funkcionalnosti koje pruža Socket.IO knjižnica započinje inicijalizacijom HTTP poslužitelja kojem se kao parametar prosljeđuje instanca Express aplikacije. Na ovaj način HTTP poslužitelju omogućava se da obrađuje zahtjeve koristeći rute i middleware funkcije definirane u Express aplikaciji. Programski kod za kreiranje i pokretanje poslužitelja koji osluškuje definirani port za moguće konekcije i omogućuje dvosmjernu komunikaciju u stvarnom vremenu prikazan je na slici 4.7.

```
import { Server } from "socket.io";
import http from 'http'
import express from 'express'

const app = express();                          //express app

const server = http.createServer(app);          //creates http server
const io = new Server(server, {                 //socket.io server based on http
    cors: {
        origin: ["http://localhost:5173"],
        methods: ["GET", "POST"]
    }
});
```

Slika 4.7. Inicijalizacija Socket.IO poslužitelja

4.3.3 Kreiranje potrebnih globalnih konteksta

Kako bi informacije vezane za ostvarene Socket.IO veze bile dostupne u cijeloj React aplikaciji (na klijentskoj strani), stvoren je React kontekst (engl. *context*) čije se stanje sastoji od 2 varijable: *socket* (koja se koristi za postavljanje Socket.IO veze prilikom prijave korisnika) i *onlineUsers* (koja sadrži popis trenutno aktivnih korisnika, a koristi se na administratorskoj strani dopisivanja). Nakon kreiranja potrebnog konteksta, definira se i

funkcija (točnije *React hook*) koja se koristi za lakši pristup ovim varijablama od strane React komponenti u kojima se realizira dopisivanje na klijentskoj strani. Ovaj kontekst koristi pomoć drugog definiranog konteksta – AuthContext za dohvat podataka autentificiranog korisnika kako bi se veza dodijeljena tom korisniku povezala s njegovim ID-em.

Koristeći se *useEffect* funkcijom koja se izvršava pri svakoj promjeni *authUser* stanja, omogućava se uspostava nove Socket.IO veze svaki put kada se korisnik autentificira. Tom prilikom, stvorenoj vezi se prosljeđuje ID korisnika kao upitni (engl. *query*) parametar i vrši se postavljanje nove vrijednosti *socket* varijable u stanje definirano kontekstom. Tom prilikom vrši se i ažuriranje liste trenutno aktivnih korisnika, ostvareno kao odgovor na događaj naziva *getOnlineUsers* kojim poslužitelj šalje tu listu. Kada se korisnik promijeni, odnosno odjavi, zatvara se i njemu dodijeljena veza funkcijom *socket.close()*. Na slici 4.8 prikazuje se dio programskog koda iz konteksta, potreban za postavljanje nove veze prilikom autentifikacije korisnika i njeno zatvaranje po njegovoj odjavi.

```
//using the custom hook to obtain authenticated user information
const { authUser } = useAuthContext();

    //hook that runs when authUser changes:
    useEffect(() => {
        if (authUser) {
            const socket = io("http://localhost:5000", {
                //if user is authenticated, create new connection + pass his ID as query param
                query: {
                    userId: authUser._id
                }
            });
            setSocket(socket);    //storing the created socket in state
            //event listener for "getOnlineUsers" event
            // updates onlineUsers when the server sends the list of online users
            socket.on("getOnlineUsers", (users) => {
                setOnlineUsers(users);
            })

            //cleanup function that will close the WebSocket connection
            //when the component unmounts or when authUser changes
            return () => socket.close();
        } else {
            //close the existing connection if user disconnects:
            if (socket) {
            socket.close();
            setSocket(null);
        }
    }
    }, [authUser]);
```

Slika 4.8. Ostvarivanje Socket.IO veze pri autentifikaciji korisnika

Osim ovog konteksta, kreiran je i kontekst razgovora (engl. *ConversationContext*) čije se stanje sastoji od dvije varijable: *selectedConversation* i *messages*. *SelectedConversation* varijabla koristi se za označavanje razgovora sa pojedinim korisnikom. Ova funkcionalnost koristi se iz administratorske perspektive dopisivanja i služi kako bi se identificirao korisnik na čiji je on razgovor kliknuo (kojem želi poslati poruku) - potom se otvara razgovor sa tim korisnikom i aplicira se dizajn koji označava da je on odabran (promjena boje pozadine). *Messages* varijabla koristi se prilikom dohvata poruka sa traženim korisnikom.

4.3.4 Povezivanje korisnika i uspostavljene Socket.IO veze

Kako bi se aktivno pratilo stanje aktivnih korisnika, na poslužiteljskoj strani stvara se objekt naziva *userSocketMap* u kojem se u obliku ključ-vrijednost parova pohranjuju ID brojevi korisnika (poslani kao upitni parametar prilikom stvaranje veze) asocirani sa ID brojem njemu pripadajuće veze. Prilikom svakog novog povezivanja i prekida veze ovaj objekt ažurira se dodavanjem, odnosno brisanjem odgovarajućeg para vrijednosti, prilikom čega poslužitelj emitira događaj pod nazivom *getOnlineUsers* putem funkcije *emit()*, kojom se taj događaj šalje svim trenutno povezanim klijentima. Kreiranje spomenutog objekta i funkcije kojima se ažurira njegovo stanje implementirani su programskim kodom na slici 4.9.

```
const userSocketMap = {};

//listen for connections:
io.on('connection', (socket) => {
    //executes every time someone connects to server
    console.log("A user connected to: ", socket.id);


    //catch the userId passed from context to save it into map:
    const userId = socket.handshake.query.userId;
    if (userId && userId != "undefined") {
        userSocketMap[userId] = socket.id;

        //since we updated the user-socket map, we emit the event to all connected clients:
        io.emit("getOnlineUsers", Object.keys(userSocketMap));
    }

    //executes every time someone disconnects
    socket.on("disconnect", () => {
        console.log("User disconnected on: ", socket.id);

    //when users disconnects, remove it from the map:
    delete userSocketMap[userId];
    //since we updated the user-socket map, we emit the event to all connected clients:
    io.emit("getOnlineUsers", Object.keys(userSocketMap));
```

Slika 4.9. Kreiranje i upotreba objekta za pohranu korisnika i njemu dodijeljene veze

4.3.5 Funkcionalnost slanja poruke (poslužiteljska strana)

Funkcionalnost slanja poruke odvija se putem za to definirane rute iz čijih se parametara može iščitati ID primatelja i pošiljatelja. Tom prilikom se najprije u bazi podataka traži razgovor između ta dva korisnika, kako bi se, ukoliko prethodno nisu ostvarili komunikaciju, stvorio novi razgovor. Potom se u bazu podataka najprije dodaje nova poruka (u kolekciju poruka), da bi se potom dodala referenca na nju u odgovarajući razgovor.

Inkorporacija Socket.IO knjižnice se tom prilikom dešava putem funkcije *emit()*, no ovoga puta se događaj imena *newMessage* šalje samo korisniku čiji je ID dohvaćen iz objekta *userSocketMap*, koristeći za to predviđenu funkciju *getReceiverSocketId*. Kod koji se odnosi na stvaranje nove poruke i njeno slanje odgovarajućem korisniku prikazano je na slici 4.10.

```
//create the newly sent message:
const newMessage =  new Message({
    senderId: senderId,
    receiverId: receiverId,
    message: message,
});

//add the message to the conversation
if (newMessage) {
    conversation.messages.push(newMessage._id);
}

//save the message and conversation data in database (runs in parallel):
await Promise.all([conversation.save(), newMessage.save()]);


//SOCKET.IO functionality
const receiverSocketId = getReceiverSocketId(receiverId);
if (receiverSocketId) {
    //send an event only to this particular user:
    io.to(receiverSocketId).emit("newMessage", newMessage);
}

return res.status(201).json(newMessage);
```

Slika 4.10. Funkcionalnost slanja poruke

4.3.6 Funkcionalnost slanja poruke (klijentska strana)

Slanje poruke sa klijentske strane odvija se u komponenti *MessageInput*. U ovoj jednostavnoj komponenti koja se sastoji od jednog input elementa i jednog gumba. Korisnik u input element unosi tekst željene poruke, a klikom na taj gumb poziva se funkcija *useSendMessage()*. Ukoliko korisnik ne unese nikakav tekst, ova funkcija se ne poziva na izvršavanje. Nakon izvršenja funkcije, vrijednost varijable *messages* iz *ConversationContext*-a ponovno se postavlja na prazan string.

Funkcija *useSendMessage()* vrši slanje zahtjeva na ranije spomenutu rutu definiranu na poslužitelju koja je odgovorna za slanje poruke. Pri tome se ID broj pošiljatelja preuzima

iz *AuthContexta*, a ID broj primatelja dohvaća iz *ConversationContexta*, a i sama poruka se dodaje u *messages* stanje tog konteksta. U slučaju da poslužitelj povratno ne pošalje podatke, odnosno dođe do pogreške, koristeći *react-toastify* knjižnicu pojavljuje se odgovarajuća obavijest. Programski kod *useSendMessage* funkcije vidljiv je na slici 4.11.

```
const useSendMessage = () => {
    const [loading, setLoading] = useState(false);
    const { authUser } = useAuthContext();
    const { messages, setMessages, selectedConversation} = useConversation();

  const sendMessage = async (message) => {
    setLoading(true);
    try {
        const res = await fetch(`http://localhost:5000/api/messages/send/${authUser._id}/to/${selectedConversation._id}`, {
            method: 'POST',
            headers: {
                'Content-Type' : 'application/json'
            },
            body: JSON.stringify({message})
        })

        const data = await res.json();
        if (data.error) {
            throw new Error(data.error);
        }

        //add the new message (data) onto the previously exchanged messages:
        setMessages([...messages, data]);

    } catch (error) {
        toast.error(error.message);
    } finally {
        setLoading(false);

    }
  }

  return { loading, sendMessage};
}
```

Slika 4.11. Funkcija namijenjena slanju poruke sa klijentske strane

4.3.7 Funkcionalnost dohvata poruke (poslužiteljska strana)

Kao i sve ostale funkcije potrebne za ostvarivanje funkcionalnosti dopisivanja, za dohvat poruke je najprije definirana odgovarajuća ruta čiji dinamički dodijeljeni parametri prenose informaciju o ID broju korisnika kojem se šalje poruka i trenutno registriranog korisnika (pošiljatelja). Po dohvatu tih informacija, pronalazi se razgovor u bazi podataka čiji su učesnici ta dva korisnika. Ukoliko takav razgovor ne postoji, vraća se prazan znakovni niz, odnosno nema prikaza poruka budući da je ovo prva interakcija između njih. Programski kod potreban za dohvat razgovora, koji kao odgovor vraća polje poruka od kojih se isti sadrži, prikazan je na slici 4.12. Kao i za ostale funkcionalnosti, logika za rad s porukama implementirana je u za to predviđenom *controlleru* kako bi se postigla fragmentacija i urednija organizacija logičkih cjelina koda.

```
export const getMessages = async (req, res) => {
    try {
        const userToChatId = req.params.userToChatId;        //user we want to chat with (see messages with them)
        const loggedInId = req.params.loggedInId;                  //currently logged in user

        //find convo between them:
        const conversation = await Conversation.findOne({
            participants: { $all: [loggedInId, userToChatId] },
        }).populate("messages");                        //to get message content, not only their id stored in messages

        if (!conversation) return res.status(200).json([]);     //if there are no messages, return empty array


        res.status(200).json(conversation.messages);

    } catch (error) {
        console.log("Error in getMessages controller: ", error.message);
        res.status(500).json({ error: "Internal server error"} );
    }
}
```

Slika 4.12. getMessages controller za dohvat poruka

4.3.8 Funkcionalnost dohvata poruka (klijentska strana)

Dohvat poruka na klijentskoj strani vrši se pozivom funkcije *useGetMessages* koja šalje zahtjev na ranije definiranu rutu za dohvat poruka na poslužiteljskoj strani. Prikaz dohvaćenih poruka odvija se u komponenti Messages, koja osim te koristi i funkciju za osluškivanje novo pristiglih poruka. Ova funkcija bazira se Socket.IO događaju pod nazivom *newMessage*, ranije spomenutog prilikom opisivanja funkcionalnosti slanja poruke sa poslužiteljske strane. Bit koda koji sačinjava ovu funkciju svodi se na to da se novo pristigla poruka doda u stanje definirano kontekstom razgovora, te da se po odspajanju ove komponente prestane osluškivati ova događaj. Korištenjem *useRef()* React hooka, omogućeno je da sve prilikom svakog pristizanja nove poruke, komponenta u kojoj je prikazan cijeli razgovor automatski pomakne prema dolje, kako bi uvijek bila vidljiva najnovije pristigla poruka. Programski kod napisan kako bi se ostvarila implementacija dohvata svih poruka sa odgovarajućim korisnikom, koji čini funkciju *useListenMessages()* vidljiv je na slici 4.13.

```
import React, { useEffect } from 'react'
import { useSocketContext} from "../src/context/SocketContext";
import {useConversation} from '../src/context/ConversationContext';



const useListenMessages = () => {
  const { socket } = useSocketContext();
  const { messages, setMessages } = useConversation();

  useEffect(() => {
    //listening for the newMessage event
    socket?.on("newMessage", (newMessage) => {
        //add this new message to messages
        setMessages([...messages, newMessage])
    })
    //when component unmounts, we don't want to still listen for this event
    return () => socket?.off("newMessage");
  }, [ socket, setMessages, messages])
}

export default useListenMessages
```

Slika 4.13. Programski kod za osluškivanje novih poruka

5 ZAKLJUČAK

AutoLoveRi web je aplikacija osmišljena za prodaju novih i rabljenih automobila. Ona služi kao virtualni izlog tvrtke specijalizirane za prodaju vozila, omogućujući potencijalnim kupcima da istraže ponudu automobila, pročitaju recenzije zadovoljnih korisnika i steknu uvid u način poslovanja iste.

Za izradu aplikacije korišten je MERN Stack razvojni okvir koji se sastoji od ukupno 4 tehnologije – MongoDB, Express, React i Node.js. Razlog odabira ovih tehnologija je prije svega želja za savladavanjem nečeg novog i sticanje novih vještina u radu sa danas sve zastupljenijim web tehnologijama. Jedna od vodećih olakotnih okolnosti u radu sa ovim tehnologijama je svakako i činjenica da sve one imaju velike i aktivne zajednice s mnoštvom dokumentacije, čime je susret sa svakom preprekom uvelike lakši za premostiti. Dobra strana je i činjenica da se njima može izgraditi *full-stack* aplikacija zasnovana na jednom glavnom programskom jeziku, a to je u ovom slučaju JavaScript. Time je vrijeme učenja i savladavanja osnovnih koncepata značajno skraćeno, a povezivanje klijentskog i poslužiteljskog dijela olakšano.

Iako je izgrađena aplikacija obimna i ispunjena mnogobrojnim funkcionalnostima, moguća su poboljšanja u pogledu bolje iskoristivosti već kreiranih komponenti ekrana. Aplikacija bi se dodatno mogla modernizirati uvođenjem samostalnog *chatbota* koji korisnicima odgovara na neka osnovna pitanja, tako da ne moraju čekati odgovor administratora. Osim toga, moguće je implementirati svojevrsni blog odjeljak stranice, putem kojeg bi se korisnici mogli informirati o novostima iz auto-moto svijeta.

LITERATURA

[1] MongoDB, s Interneta, https://www.mongodb.com/, 10. travanj 2024.

[2] MongoDB Atlas, s Interneta, https://www.mongodb.com/atlas, 10. travanj 2024.

[3] Mongoose, s Interneta, https://mongoosejs.com/, 10. travanj 2024.

[4] BcryptJS, s Interneta, https://www.npmjs.com/package/bcryptjs, 14. travanj 2024.

[5] Bcrypt hashing algoritam, s Interneta, https://en.wikipedia.org/wiki/Bcrypt, 14. travanj 2024.

[6] Node.js, s Interneta, https://nodejs.org/en, 21. travanj 2024.

[7] Express.js, s Interneta, https://expressjs.com/, 21. travanj 2024.

[8] cookie-parser, s Interneta, https://www.npmjs.com/package/cookie-parser, 21. travanj 2024.

[9] cors, s Interneta, https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS, 21. travanj 2024.

[10] React, s Interneta, https://react.dev/, 23. travanj 2024.

[11] React hooks, s Interneta, https://react.dev/reference/rules#rules-of-hooks, 23. travanj 2024.

[12] TailwindCSS, s Interneta https://tailwindcss.com/, 25. travanj 2024.

[13] jsonwebtoken, s Interneta

https://www.npmjs.com/package/jsonwebtoken, 2. svibnja 2024.

[14] Socket.IO, s Interneta

https://socket.io/, 25. svibanj 2024.

[15] WebSockets, s Interneta,

https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, 25. svibanj 2024.

[16] AOS, s Interneta,

https://michalsnik.github.io/aos/, 23. travanj 2024.

[17] react-icons, s Interneta,

https://react-icons.github.io/react-icons/, 23. travanj 2024.

[18] react-toastify, s Interneta,

https://www.npmjs.com/package/react-toastify, 24. travanj 2024.

[19] PayPal developer, s Interneta.

https://developer.paypal.com/build-better/, 3. svibanj 2024.

[20] @paypal/react-paypal-js, s Interneta,

https://www.npmjs.com/package/@paypal/react-paypal-js, 3. svibanj 2024.

[21] multer, s Interneta,

https://www.npmjs.com/package/multer, 5. svibanj 2024.

[22] daisyUI, s Interneta,

https://daisyui.com/, 20. svibanj 2024.

POPIS SLIKA

SAŽETAK

AutoLoveRi je web aplikacija namijenjena prodaji novih i rabljenih automobila. Ova platforma predstavlja digitalni izlog kompanije koja se bavi prodajom vozila, omogućujući korisnicima da pregledaju dostupnu ponudu, pročitaju recenzije drugih kupaca i steknu uvid u način poslovanja. Korisnici mogu jednostavno pretraživati, filtrirati i sortirati automobile, čime se olakšava i ubrzava pronalazak željenog vozila. Nakon odabira, vozilo se može kupiti online putem PayPala. Dodatno, aplikacija omogućava korisnicima da se u realnom vremenu dopisuju s administratorima za dodatne upite ili provjeru statusa narudžbe. Aplikacija je izrađena pomoću MERN Stack razvojnog okvira koji uključuje MongoDB, Express.js, React i Node.js, a za izradu je dodatno korišten i Tailwind CSS za vizualni dizajn. Jedna od najvećih prednosti ovog razvojnog okvira je omogućavanje dosljednog razvoja jer sve komponente koriste JavaScript za kodiranje i na poslužiteljskoj i na klijentskoj strani.

Ključne riječi: Web, aplikacija, MERN, MongoDB, Express, React, Node.js, Socket.IO, TailwindCSS

ABSTRACT

AutoLoveRi is a web application intended for the sale of new and used cars. This platform is a digital showcase of a company that sells vehicles, allowing users to view the available offer, read reviews of other customers and gain insight into business principles of the company. Users can easily search, filter and sort cars, making it easier and faster to find the desired vehicle. Once selected, the vehicle can be purchased online via PayPal. In addition, the application allows users to correspond in real time with administrators for additional inquiries or in case they need to check the status of the order. The application was created using the MERN Stack development framework, which includes MongoDB, Express.js, React and Node.js, with addition to Tailwind CSS which was used for UI design. One of the biggest advantages of this development framework is that it enables consistent development because all components use JavaScript for both server-side and client-side coding.

Keywords: Web, Application, MERN, MongoDB, Express, React, Node.js, Socket.IO, TailwindCSS