Izrada web aplikacije za stvaranje i upravljanje radnim nalozima

Ćavar, Luka

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetna Informatika

Luka Ćavar

Izrada web aplikacije za stvaranje i upravljanje radnim nalozima

Završni rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, 3.9.2021

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 20.4.2021.

Zadatak za završni rad

Pristupnik: Luka Ćavar

Naziv završnog rada: Izrada web aplikacije za stvaranje i upravljanje radnim nalozima

Naziv završnog rada na eng. jeziku: Development of a web application for creating and managing work orders

Sadržaj zadatka: Glavni zadatak završnog rada je izraditi web aplikaciju za stvaranje i upravjanje radnim nalozima. Pritom će se koristiti odabrani alati za razvoj frontend i backend dijela aplikacije koji će se u radu i detaljno opisati. Također će se opisati i demonstrirati svi važni elementi i funkcionalnosti izrađene web aplikacije. U radu će se također pojasniti razlozi za korištenje aplikacije za stvaranje radnih naloga te prednosti koje njeno korištenje donosi.

Mentor

Doc. dr. sc. Lucia Načinović Prskalo

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 16.4.2021.

(potpis pristupnika)

Sadržaj

Sažetak

Web aplikacijama smatraju se računalni programi koji koriste web tehnologiju i web preglednike za omogućivanje korisnicima slanje i dohvaćanje podataka iz neke udaljene baze podataka, odnosno poslužitelja. Web aplikacija *Radni nalozi* namjenjena je za stvaranje i upravljanje radnim nalozima. Radni nalozi dokumenti su koji sadrže informacije i detalje o određenom poslu poduzeća te sve potrebne dokumentacije kako bi se zahtjev odobrio. Cilj ovog završnog rada je digitalizirati unos podataka u radne naloge te olakšati i ubrzati postupak izrade istog. Izradom web aplikacije automatizirana je pohrana radnih naloga i podataka koji oni obuhvaćaju.

U izrađenoj aplikaciji korištišteni su sljedeći alati ili skupine alata: Visual Studio Code kao razvojno okruženje, Quasar kao softverski razvojni okvir te programski jezici Vue.js, Node.js, JavaScript.js, HTML/CSS. Svaki je od tih alata i njihova primjena u izađenoj aplikaciji pojašnjena i opisana. Također su detaljno opisani svi elementi aplikacije i njihove funkcionalnosti.

Ključne riječi: Web aplikacije, Radni nalog, Baza podataka, Visual Studio Code, Vue.js, Node.js, Javascript, HTML, CSS

1. Uvod

1.1 Web aplikacije

Web aplikacije su računalni programi koji koriste web tehnologiju i web preglednike kako bi omogućili korisnicima slanje i dohvaćanje podataka iz neke udaljene baze podataka. Jednostavnije rečeno, to su programi koji omogućuju bolju komunikaciju između tvrtki i njihovih klijenata. Prema uredniku Web.AppStorm Jarelu Remicku, *„svaka komponenta web stranice koja izvršava neke funkcije za korisnika kvalificira se kao web aplikacija“*[1]. Jedna od najvažniji dobrobiti web aplikacija jest da mogu biti dostupne za korištenje svima, ovisno o klijentima i njihovim potrebama. Prema tome razlikujemo više vrsta web aplikacija, a neke od njih su web aplikacija za e-trgovinu, web aplikacija portala, SPA (eng. Single Page Application) web aplikacije poput Netflixa i Gmaila, PWA (eng. Progressive web application) web aplikacije kao što je Forbes te druge.

1.1.1 Kako web aplikacija funkcionira

Web aplikacije napisane su programskim jezikom kojeg podržava neki web preglednik (najčešće Javascript i HTML). One sadrže model klijent-poslužitelj koji se sastoji od razvoja klijentskih i poslužiteljskih računalnih aplikacija. Klijentska aplikacija je aplikacija koja se izvodi unutar internetskog preglednika, tj. od strane klijenata koji koriste aplikaciju. Pisanje programskog koda odvija se u jezicima kao što su HTML/CSS, JavaScript i drugi, a ovu stranu aplikacije nazivamo FrontEnd ili sučelje koje se prikazuje korisnicima/klijentima. S druge strane, poslužitelj aplikacija izvodi se na nekom udaljenom poslužitelju ili serveru koji obavlja zadatke za klijente, a njega programer (osoba koja piše programski kod) piše u jezicima kao što su Python, PHP i drugi te ovu stranu aplikacije nazivamo BackEnd [2].

Rad web aplikacija možemo opisati u nekoliko koraka:

1. Korisnik preko korisničkog sučelja aplikacije šalje zahtjev web poslužitelju putem interneta
2. Poslani zahtjev web poslužitelj prosljeđuje poslužitelju web aplikacija
3. Traženi zadatak (npr. obrada podataka) se izvršava od strane poslužitelja web aplikacija te se rezultati traženih zadataka stvaraju/generiraju
4. Poslužitelj web aplikacija šalje te rezultate natrag web poslužitelju (obrađeni podaci)
5. Web poslužitelj zatim šalje klijentu tražene podatke koji se nakon toga pojavljuju na korisničkom sučelju

1.1.2 Uporaba web aplikacija

Web aplikacije imaju široku upotrebu. Ljudi diljem svijeta ih svakodnevno koriste prilikom online kupovine ili slanjem e-maila (Gmail, Yahoo) . Također, sveopće su prisutne u poslovnim tvrtkama te postoje web aplikacije specifirane za procesore teksta, proračunske tablice, skeniranje i pretvaranje datoteka i slično. S druge strane, web aplikacije se koriste kao platforme za gledanje filmova i serija (Netflix) ili kao internet bankarstvo, online aukcije ili u sklopu web dućana (plaćanje, kolica za kupovinu)[3].

1.1.3 Motivacija, značaj i prednosti web aplikacija

Web aplikacije imaju veliki značaj u svakodnevnici. Sve više su uključene u svaki dio našeg, kako poslovnog, tako i društvenog života.
U poslovnom smislu, web aplikacije bitne su tvrtkama iz više razloga. Jedan od razloga je taj što web aplikacije pomažu kod publiciteta i brendiranja tvrtki. Pomoću njih je lakše održati komunikaciju sa potencijalnim klijentima i s trenutnim korisnicima preko korisničke podrške. Omogućuju isticanje svojih proizvoda i prodaju usluga kroz web aplikacije te samim time i širenje svoga brenda kroz internet. Time što tvrtke sve više ulažu u razvoj svojih web aplikacija dolazi do konkurentnosti na tržištu te i druge tvrtke koje su specificirane za tu granu proizvodnje imaju također koristi. U društvenom smislu, web aplikacije omogućile su nam nesmetano gledanje i slušanje zabavnog programa te lakšu komunikaciju između ljudi. Određene aplikacije prikazuju događaje koje su u području gdje se korisnik nalazi i predlažu one koje bi korisnika mogli zanimati. Također, aplikacije mogu olakšavati navigaciju na putovanjima(primjerice Google Maps).
Prednosti web aplikacija su da mogu raditi na više platformi i uređaja (računalo, mobitel, tablet) sve dok je preglednik kompatibilan [3]. Aplikaciju nije potrebno instalirati na tvrdi disk računala. Aplikacijama se može pristupiti bilo gdje s web preglednikom te korisnici pristupaju istoj univerzalnj verziji aplikacije čime se otklanja problem kompatibilnosti. Ujedno je pristup dostupan u bilo koje vrijeme, neovisno o radnom vremena tvrtke ili klijenata. Olakšavanje poslovanja je bitna prednost jer web aplikacije smanjuju trošak i za tvrtku i za korisnika na način da je potrebno manje održavanja i podrške od strane tvrtke te niže tehničke zahtjeve računala klijenta. Također, web aplikacije mogu imati veliku razinu sigurnosti ukoliko se radi o aplikacijama koje funkcioniraju na naplati usluga preko interneta (primjerice pretplata Netflixa). Web aplikacija pohranjuje svoje podatke u oblak, tako da se u slučaju gubitka podataka ili nekakve pogreške, podaci mogu jednostavno vratiti iz oblaka. Poboljšana učinkovitost kao jedna od prednosti korištenja web aplikacija očituje se u automatiziranosti - veća količina zadataka se obavlja u manje vremena što je vrlo bitno u poslovnom smislu. Ljudske greške su svedene na minimum i time se ranjivost sustava uklanja. Pošto se web aplikacije proizvode „po mjeri“ klijenta, time su napravljene prema određenim zahtjevima kako bi poslovanje bilo što jednostavnije, preciznije i efikasnije. Ovime se korisnik ne treba prilagđavati nekom gotovom proizvodu i trošiti vrijeme na učenje korištenja softvera. Ako bi poslovanje raslo ili se promijenilo u nekom smislu, web aplikacija može biti modificirana prema novim zahtjevima i potrebama korisnika/klijenta.

1.1.4 Rast i budućnost web aplikacija

Rast web aplikacija uzrokovan je sve većom potrebom za pojednostavljenjem svakodnevnih poslova. One su promijenile svaki aspekt našeg života. U članku [4] autor prikazuje razvoj i rast web aplikacija u budućnosti kroz nekoliko trendova koji bi mogli unaprijediti i poboljšati naše iskustvo. Neki od trendova su: progresivne web aplikacije (PWA), umjetna inteligencija (AI), internet stvari (IoT), glasovne komande vezane za navigaciju i trgovinu i drugi.

Progresivne web aplikacije (Progressive Web Application – PWA) ili *„Aplikacije specifične za platformu osjećaju se kao dio uređaja na kojem rade“* [5] su web aplikacije koje koriste napredne tehnologije kako bi mogli iskoristiti prednosti web i izvornih aplikacija. Poznate su po svojoj pouzdanosti i bogatom repertoaru zadataka koje mogu ispuniti. Progresivne aplikacije mogu funkcionirati i bez spajanja na mrežnu vezu. Na njima je, primjerice, moguća produkcija zabavnog sadržaja u pozadini dok koristite drugu aplikaciju. Neke od najpoznatiji web aplikacija kao što su Youtube, Twitter ili Hulu su zabilježile porast korištenja svojih platformi nakon pokretanja svojih novih progresivnih web aplikacija. Umjetna inteligencija (Artificial Intelligence – AI) se sve više primijenjuje pri izradi web aplikacija. Osim novčanog prihoda koja će prema statistici [4] do 2025. godine dostići preko 1.25 milijardi američkih dolara, AI je sveprisutan u najpoznatijim svjetskim tvrtkama kao što su IBM, Google, Microsoft i druge. Neke od najvažnijih značajki koje krase web aplikacije su: prepoznavanje glasova i lica, kvantno računarstvo i računarstvo u oblaku, chatboti i korisnička podrška pomoću AI. Chatbotovi nude zamjenu ljudske podrške pomoću tehnike prepoznavanja govora i kognitivne inteligencije. Oni nude interakciju s ljudima pomoću obrade prirodnog jezika. Jedan od poznatijih primjera je Swelly, Facebook Messenger chatbot.

Internet stvari (Internet Of Things – IoT) koriste se sve češće u web industriji. Pojam IoT odnosi se na sve fizičke stvari koje su spojene na Internet, a prikupljaju i dijele podatke. Temelje se na povezivanje različitih objakata sa senzorima koji omogućuju komunikaciju sa podacima pomoću aplikacija, čime se olakšava manipulacija nad objektima i pojednostavljuje proces. Primjerice, žarulja se može uključiti ili kućni sigurnosni sustav se može upravljati preko aplikacije na mobilnom telefonu.

1.1.5 Značaj web aplikacija u doba pandemije

Posljednjih je mjeseci cijeli svijet zahvatila pandemija uzrokovana virusom, a to je osim na zdravlje ljudi utjecalo i na brojne druge aspekte pa tako i na ekonomije i gospodarstva država u vidu njihova slabljenja. Web aplikacije su pak u to vrijeme rasle i razvijale se. Kako je društvo bilo primorano ostati kod kuće i raditi od doma, tako su tvrtke razvijale svoje proizvode dostupne svima i svugdje. Aplikacije poput Zooma pomogle su ljudima obavljati svakodnevne poslove, kako poslovne tako i društvene. Komunikacija koja je prije bila uživo, prebacila se na web aplikacije.. Tvrtke i organizacije su pomoću takvih aplikacija mogle održavati poslovne sastanke. Online plaćanje je olakšalo kupovinu proizvoda te su se time razvijale i web računovostvene aplikacije u online poslovanju tvrtki [6]. Sigurnost je postala jako važna jer su ljudi sve više koristili i spremali bitne dokumente na svoja računala. Ovdje su ulogu preuzele web aplikacije za pohranu na oblaku.

2. Web aplikacije za stvaranje i upravljanje radnim nalozima

Web aplikacije za stvaranje i upravljanje radnim nalozima su računalnim programi koji opisuju, stvaraju i upravljaju službenim dokumentima koji opisuju zahtjev za odobrenjem nekog službenog rada/posla. Općenito, radni nalozi sadrže informacije i detalje o određenom poslu te sve potrebne dokumentacije kako bi se zahtjev odobrio [23]. Nakon što se zahtjev pregleda i odobri, izdaje se službeni radni nalog za dovršetak posla.

2.1 Prednosti i korištenje web aplikacija za stvaranje i upravljanje radnim nalozima

Web aplikacije za stvaranje i upravljanje radnim nalozima koriste se u poslovnim tvrtkama. Njihova glavna uloga jest objasniti i opisati posao koji je potrebno obaviti. Ljudski faktor u ovom slučaju predstavlja veliki značaj. Naime, pri stvaranju radnih naloga potrebna je pedantnost i nema prostora za pogreške [7]. Uz ovakvu vrstu sofvera to se otklanja jer aplikacija nudi veliku širinu detalja koji ljudskom oku lako promaknu. Iz tog razloga, stvorena je web aplikacija koja je specificirana kako bi olakšala stvaranje i upravljanje radnih naloga. Za početak, aplikacija automatski bilježi potrebne informacije čime se uklanja potreba za ručnim unosom. Primjerice, bilježi trenutno vrijeme početka pravljenja ili nastanka samog radnog naloga. Također, neke takve aplikacije imaju komponentu koja prati kako napreduje izrada samog naloga - *status*, koji izvještava korisnika u kojoj je fazi izvršenja dani nalog. Informacije o klijentima i poduzeću uvijek su vidljive i dostupne, te se sortiranje dokumenata može lako izraditi ovisno o traženom zahtjevu korisnika [8].

2.2 Važnost web aplikacija za stvaranje i upravljanje radnim nalozima

Web aplikacije specifirane za radne naloge važne su za svaku tvrtku, neovisno o vrsti njezinog posla. One omogućuju jasniji i bolji pregled nad dokumentima, veću organiziranost i učinkovitost. Također, takve aplikacije štede vrijeme i troškove, podaci i informacije se ažuriraju u stvarnom vremenu te se mogu koristiti i slati s jednog na drugo računalo neovisno o lokaciji korisnika. Sve navedene prednosti važne su tvrtkama i organizacijama kako bi olakšale svoje poslovanje i dobro funkcionirale, što je najvažniji razlog postojanja aplikacije za stvaranje i upravljanje radnim nalozima.

2.3 Specifičnost web aplikacija za stvaranje i upravljanje radnim nalozima

Specifičnost web aplikacija za stvaranje i upravljanje radnim nalozima očituje se u njezinoj raznolikosti. Postoji više vrsta aplikacija specifiranih upravo za radne naloga, a one pokrivaju široki raspon različitih vrsta poslova koji se moraju obaviti. Pa tako, primjerice postoje radni nalozi inspekcijske prirode, hitni radni nalozi, radni nalozi vezani uz sigurnost, radni nalozi za električne kvarove, preventivni radni nalozi i sl. [9] U aplikacija za inspekciju, radni nalozi su usredotočeni u pravom smislu riječi na isnpekciju vaše imovine, tvrtke ili neke druge organizaciju u cilju pregleda svih dokumenata kako bi se utvrdilo jesu li

valjani. Hitni softveri specificirani su za slučajeve u kojoj tvrtka ili organizacija osigurava svoju imovinu od požara, poplava i drugi nepogoda [9]. Sigurnosti softveri zaduženi su za smanjivanje sigurnosnih i zdravstvenih rizika dok su električni softveri zaduženi za sprječavanje bilo kakve nezgode uzrokovanje rasvjetom, napajanjem ili ožičenjem. Preventivni softver bavi se rutinskim poslovima vezano za određene strojeve ili mašine gdje se, nakon svakog period, provjeravaju isprvnosti određenih komponenti radi osiguravanja ispravnosti rada stroja. Također, ti softveri se zovu preventivni kako bi uklonili vjerojatnost nastanka kvara uzrokavano bilo kakvim vanjskim čimbenikom.

3. Opis postojećih softvera za stvaranje radnih naloga

Na tržištu postoji mnogo dostupnih aplikacija za pomoć pri poslovnim procesima, a tako i za stvaranje i upravljanje radnim nalozima. Njihova uloga je pružanje potpore tvrtkama i organizacijama u poslovanju, čime ono postaje učinkovitije i lakše. Neki od poznatijih primjera takvih aplikacija prema stranici capterra.com [10] će se pobliže opisati u sljedećim poglavljima.

3.1 Wrike

Wrike [11] je softver za upravljanje i praćenje projekta te radnih naloga. Pojednostavljenje rada poslovanja kako bi se tvrtka mogla usredotočiti na osnovne zadatke glavni je cilj ovog softvera. Također, suradnja unutar tvrtke je vrlo važna i upravo zbog tog razloga ovaj softver olakšava svaki sastavak i razgovor kako bi korisnici mogli podatke organizirati u rezultate i planove. Ukoliko su potrebni dodatni dogovori, za to je dostupan i vlastiti chat. Zahtjevi se brzo analiziraju i odobravaju jer Wrike omogućuje korisnicima detaljan prikaz svih informacija koje su potrebne kako bi se došlo to brzog zaključka. Omogućena je potpuna transparentnost nad svakim radnim nalogom gdje korisnik može u slučaju zastarjenja naloga promijeniti ili nadodati novi zadatak, njegov opis ili drugu varijablu.Također su podržani personalizirani nalozi u svrhu lakše preglednosti i izvršavanja zadataka, zajednički kalendari, organizacija informacija po prostorijama, poslovima ili drugim značajkama koje su korisniku ili timu bitni radi lakšeg poslovanja. Softver omogućuje integraciju više platformi unutar jedne. Primjerice, korisnik može izabrati web računodstvene platforme i s njima integrirati platformu za ispis nekog dokumenta ili računa ukoliko je potrebno. Svaki korisnik ima svoj ključ za pristup podacima te je svaki podatak dodatno ekriptiran ključem za pristup. Time se postigla visoka razina sigurnosti. Svi podaci i informacije su spremljeni na platformu za spremanje podataka (oblak), što osigurava podatke u slučaju kvara.

3.2 UpKeep

UpKeep [24] je tvrtka koja se specijalizirala za više vrsta radnih naloga. Razvijena su softverska rješenja koja pomažu korisnicima u više aspekata života. Oni su specijalizirani za: preventivno održavanje, upravljanje i održavanje zalihama i imovinom te radne naloge.
Preventivne web aplikacije za radne naloge usredotočene su na sitne poslove koje svakodnevno obavljamo. Upravo zato, ovo rješenje pomaže kako bi sve funkcioniralo na dobar način i kako bi se uklonile nepotrebne poteškoće. Mogućnost postavljanja vremenskog perioda nakon kojeg korisnik želi

da mu dođe obavijest o statusu nekog elementa poslovanja ili jednostavno primjerice automobila, samo je jedna od odlika ovog softvera. Omogućena je i izrada prilagođenih popisa za sigurnosne provjere, održavanja ili sličnih aktivnosti gdje se u svakom trenutku može pristupiti radnom nalogu i vidjeti detaljan plan za svaku aktivnost, njezinu starost ili neki drugi parametar.
Prema Američkom ministarstvu energetike, tvrtke mogu uštedjeti u prosjeku od 12 do 18 posto troškova ulaganjem u preventivno održavanje[12].
Softver za upravljanje imovinom omogućuje korisnicima da budu u toku sa statusom svoje imovine, od kuće do automobila. Nadzorna ploča softvera nudi preglednost nad performansama imovine, ukoliko je došlo do nekog kvara, vrši proračun koliko je vremena prošlo od prošlog održavanja, otkriva uzrok kvara po varijabli koja nije ažurirana; primjerice, u sustavu za održavanja dijelova automobila postoji varijabla *ulje* koju treba ažurirati svaki put kada se provjeri stanje ulja. Uvedeni su QR kodovi kako bi se olakšao brz pristup cijeloj potrebnoj dokumentaciji. Također, sve je transparentno i korisnik ima pregled i uvid u to tko je u kojem trenutku napravio izmjenu nad nekom komponentom što ujedno olakšava i komunikaciju.
Softver za radne naloge jednostavan je za korištenje, čime se uklanja vrijeme koje je potrebno da korisnici nauče rukovati s njime. Svi uneseni podaci koriste s e u svrhu statističke analize te time omogućuje prikaz svih mogućih problema ili prednosti kako bi korisnik bio u tijeku razvoja situacije. Softver primjenjuje koncepte Internet Stvari (IoT) pri čemu pomoću senzora može prikazati informacije iz okoline ukoliko je to potrebno.

2.3 EZOfficeInventory

EZOfficeInventory [13] je softver koji se bavi upravljanjem imovine, održavanjem i korisnicima. Softver ima mogućnost praćenja napretka radnog naloga i bilježenje povijesti usluga radi preglednosti (dnevnici napretka). Označava rizike u softveru i upozorava o rutinskim uslugama radi prevencije kvarova ili pogreški. Izvješća se izrađuju po svim parametrima koji su uneseni te se podaci prikazuju u tablicama. Također, korisnik može upravljati i pregledavati informacije o imovini na jednom mjestu. Svi podaci se ažuriraju u stvarnom vremenu. Upravljanje sigurnosti se obavlja na način da korisnici imaju pristup podacima po svojim poslovnim preferencama te postoji kontrola pristupa unutar sustava [13]. Proces skeniranja osobnog dokumenta omogućuje pristup određenim podacima vezanih za imovinu korisnika koji se prikazuju detaljno nakon autentifikacije skenom. Znatno smanjuje troškove poslovanja tvrtki i organizacije koje ga koriste te je razvijen za veliki broj industrija. Graditeljstvo, obrazovanje ili zdravstvena zaštita su neki od područja u kojem se ovaj softver koristi.

4. Opis tehnologija koje su korištene u izrade web aplikacije Radni nalozi

Kako bi se jedna web aplikacija dizajnirala i izradila, potrebni su određeni alati i znanje njegovog primjenjivanja. Pri izradi web aplikacije za stvaranje i upravljanje radnim nalozima koristio sam alate koji su opisani u sljedećim poglavljima.

4.1 Alati i programski jezici za razvoj sučelja

Alati za razvoj web aplikacija su softveri koji olakšavaju kodiranje ljudima koji pišu kod – programerima. Prilikom izrade web aplikacije uobičajeno je korištenje sljedećih alata i skupina alata: integrirano razvojno okruženje, softverski razvojni okvir i programske jezike s kojima se piše kôd softvera. U zadanoj aplikaciji koristili smo redom: Visual Studio Code kao razvojno okruženje, Quasar kao softverski razvojni okvir te programske jezike Vue.js, Node.js, JavaScript.js, HTML/CSS.

Razvojno okruženje ili IDE (engl. *Integrated Development Environment*) vrsta je računalnog softvera koji programerima služi za pisanje programskog kôda [25]. Jedna od značajki razvojnih okruženja je, uz to što se koriste za pisanje programskog kôda, da također nude i automatsko završavanje naredbi te pronalaženje pogrešaka u kodu programa. To znači da kada korisnik počne pisati naredbu, ono nudi izbor naredbi koje sadrže ta slova i automatski završava naredbu. Isto tako ako korisnik napravi neku grešku pri pisanju, softver će ga upozoriti i ukazati mu gdje je greška nastala. Visual Studio Code je razvojno okruženje korišteno u izradi web aplikacije za izradu radnih naloga. To je besplatni alat za pisanje programskog koda koji se izgrađen na principu otvorenog koda. Nudi velik izbor proširenja što znači da se u njemu može pisati kôd pomoću različitih programskih jezika.

Softverski razvojni okvir je platforma za razvoj računalnih aplikacija koja sadrži gotove programske kodove za korištenje od strane programera, a u kojoj se nalaze sve opće funkcionalnosti za razvoj aplikacija [26]. Njihova primjena pojednostavljuje i ubrzava razvoj softvera iz razloga što uključuje veliki broj funkcionalnosti. Primjerice, funkcionalnosti koje se odnose na interakciju sa klijentima poput prikaza dijaloga, padajućih lista, alatne trake, poruke korisniku i slično. Bez ovih uključenih funkcionalnosti programer bi morao sam sve pisati i stvoriti i to je upravo razlog zašto ga programeri primjenjuju. Osnovne karaktistike softverskih razvojnih okvira su: različiti programski jezici imaju svoje softverske razvojne okvire pošto su međusobno vezani; unaprijed je definirano ponašanje softverskih razvojni okvira koje programer može prilagoditi; tijek izvođenja računalne aplikacije je pod kontrolnom upravljanja softverskog razvojnog okvira; dodavanjem vanjske knjižnice moguće je proširiti softverski razvojni okvir. Softverski razvojni okvir koji je korišten u izradi web aplikacije za izradu radnih naloga je Quasar - okvir otvorenog kôda temeljen na Vue.js-a (softverski razvojni okvir za programski jezik JavaScript) [14]. Quasar je okvir koji se fokusira na performanse, lako je prilagodljiv i proširiv. Pruža podršku u svim načinima gradnje aplikacija: PWA, SPA (engl. Single Page Application), mobilnih, stolnih i ostalih. Također, ima široki raspon podrške za platforme (Google Chrome, FireFox, Opera i dr.).

Programski jezici su alati koje koristi programer kako bi napisao skup uputa koje računalo/aplikacija treba slijediti [15]. Postoji veliki broj programski jezika no ono što ih razlikuje je njihova sintaksa. Kako bi programer znao raditi pomoću programskih jezika i pisati kôd, treba naučiti njihovu sintaksu, strukturu i pravila. Programski jezici se danas primijenjuju u raznim granama, a koriste se u razvoju: aplikacija i programa, umjetne inteligencije, baza podataka, igara, Interneta i web aplikacija i dr. Programski jezici koji su korišteni u izradi web aplikacije za izradu radnih naloga su Javascript, HTML i CSS. JavaScript je programski ili skriptni jezik koji omogućuje primjenu funkcija različite složenosti na web stranice ili aplikacije [15]. U početku je JavaScript imao drugačiji naziv: LiveScript. No, s obzirom da je u vrijeme LiveScripta njegova preteča Java bila vrlo popularna, tim programera iz tvrtke Netscape odlučili su da će ime Java pomoći u popularizaciji novog programskog jezika nazvanog JavaScript. Kako se JavaScript s vremenom razvijao, postao je potpuno neovisan jezik sa vlastitom specifikacijom ECMAScript te sada uopće nema nikakvu poveznicu s jezikom Java. Mogućnosti ovog programskog jezika ovise o razvojnom okruženju u kojem se izvodi. U primjeru web aplikacije za manipulaciju radnih

naloga gdje smo koristili razvojno okruženje Node.js, JavaScriptu su omogućene funkcije za čitanje i pisanje datoteka, izvršavanje zahtjeva i slično. U pregledniku, JavaScriptom se mogu ivršavati razne funkcije a neke od najvažniji su manipulacija web stranicom te interakcija s web poslužiteljom i korisnikom. Također, u pregledniku je moguće dodavanje nove HTML datoteke stranici, mijenjanje sadržaja, preuzimanje i učitavanje datoteka sa udaljenog poslužitelja, kontrola kolačića, lokalna pohrana podataka o prijavi i drugo. Hyper Text Markup Languange ili kraće HTML jezik je koji se koristi pri izradi web stranice ili web aplikacije. „HyperText“ označava tekst koji ima vezu u sebi i svaki put kad se klikne veza, ona vodi na novu web stranicu. „Markup“ označava niz oznaka koji tumači o stilu i strukturi pisanog koda i dokumenta [16]. On se primjenjuje na tekstualnim dokumentima kako bi ih „osposobio“ i napravio ih vidljivim webu. HTML je vrlo razumljiv i jednostavan jezik; posjeduje veliki broj oznaka za oblikovanje; ne ovisi o platformi pa se može prikazati na bilo kojoj (Windows, Linux i dr.); ima mogućnost dodavanja grafike, slika i videa što ga čini privlačnim i interaktivnim. CSS ili „Cascading Style Sheet“ je stilski jezik koji se koristi za oblikovanje i opisivanje izgleda dokumenata napisanih u HTML-u. Koristi se zajedno s HTML-om za izradu web stranica, pri čemu je CSS usredotočen na izgled [16]. Pogodan je jer se stvara stil za svaku klasu unutar HTML dokumenta čime se štedi vrijeme jer nije potrebno, primjerice, za svaki naslov posebno stavljati značajke stila nego se svi naslovi obuhvate jednom klasom.

4.2 Baza podataka

Baza podataka organizirana je zbirka informacija ili podataka pohranjenim u računalnom sustavu[18]. Podaci su najčešće organizirani u tablice i stupce čime se postiže jednostavan i efikasan pregled. Njima se može pristupiti, upravljati, ažurirati mijenjati te brisati unutar baze podataka. Sustav za upravljanje bazom podataka ili kraće DBMS (engl. Database Management System) je softver koji služi za interakciju s korisnicima; on izvlači informacije iz baze podataka kao odgovor na tražene upite. Pojam baze podataka često označava isto značenje kao i sustav za upravljanje bazom podataka. Postoji više vrsta baza podataka o neke od njih su: relacijske, distribuirane, baze podataka u oblaku, objektno orijentirane i druge.

Relacijske baze podataka organiziraju podatke unutar baze podataka u tablice. Tablice se se mogu međusobno povezati na temelju zajedničkih podataka [18]. To omogućuje postojanje odnosa između različitih tablica čime se postiže bolje komunikacija i povezanost te se olakšava pregled podataka. Također, relacijske baze podataka smanjuju redundatnost – suvišnost podataka. Primjerice, nema potrebe za postojanjem podataka o jednom klijentu na više od jednog mjesta.

Distribuirane baze podataka su baze podataka koje su raspoređene na više računala, mreže računala ili web sjedišta. Podaci nisu na jednom mjestu te se u slučaju zahtjeva, distribuiraju različitim poduzećima ili organizacijama [19]. Postoje dva procesa na kojim su temeljene distributivne baze podataka, a to su replikacija i fregmentacija. Replikacija je proces u kojem sustavi pohranjuju kopije podataka na različitim mjestima. U slučaju da je cijeli baza podataka dostupna na više mjesta za skladištenje baze podataka (računalo, mreža ili web), tu se bazu podataka smatra suvišnom. Prednost ovog procesa je što povećava dostupna podataka na različitim mjestima, no s druge strane proces zahtjeva veliku ažurnost podataka i računalnih resursa. Fregmentacija je proces u kojem su odnosi fregmentirani, tj. podijeljeni u manje dijelove. Svaki od fregmenata/dijelova podijeljen je na različim mjesti gdje je potreban. Prednost ovog procesa je dosljednost podataka jer nema kopije podataka. Baze podataka u oblaku su one baze podataka koje temelje svoje usluge u oblačnom okruženju koje

može biti privatno, javno ili hibridno [20]. Sadrži mnoge isti funkcije kao i tradiciionalna baza podataka, no uključuje i dodatke pomoću računarstva u oblaku. Najveća prednost ove vrste baza podataka što su svi podaci spremljeni u oblaku, čime nema potrebe za kupovinom fizičkog hardvera. Također, njima može upravljati korisnik te im se pristupa preko web sučelja.
Za izradu web aplikacije za stvaranje i upravljanjem radnim nalozima korištena je Firestore Cloud baza podataka. Firestore Cloud je Google-ova baza podataka zasnivana na NoSQL-u koja se nalazi u oblaku. NoSQL [27] je baza podataka koja umjesto tablične strukture smješta podatke u jednu strukturu podataka, poput JSON dokumenta. Prednost ove baza podataka je primijenjivanje automatskog skaliranja; bez obzira na veličinu baze podataka, iste su performanse što znači da će proces izvođenje funkcija nad podacima u vrlo velikoj i vrlo maloj bazi podataka biti obavljen jednakom učinkovitošću. Isto tako, korisnici mogu koristiti bazu podataka izvan mreže, a sinkronizacija se događa kada se poveže veza. Nude veliku razinu sigurnosti nad podacima sa mogućnošću replikacije na različita mjesta kako bi se izbjegao njihov gubitak. Pristup svakom korisniku može se ograničiti s obzirom na identitet, uzorke i slično. Autentifikacija korisnika unutar aplikacije *Radni nalozi* integrirana je s Firebase Authentication radi jednostavnijeg postavljanja i provjere pristupa bazi podataka. Podaci se pri pohrani mogu organizirati u zbirke i dokumente. Struktura je hijerarhijska kako bi se lakše povezani podaci dohvaćali i čitali.

4.3 Hosting

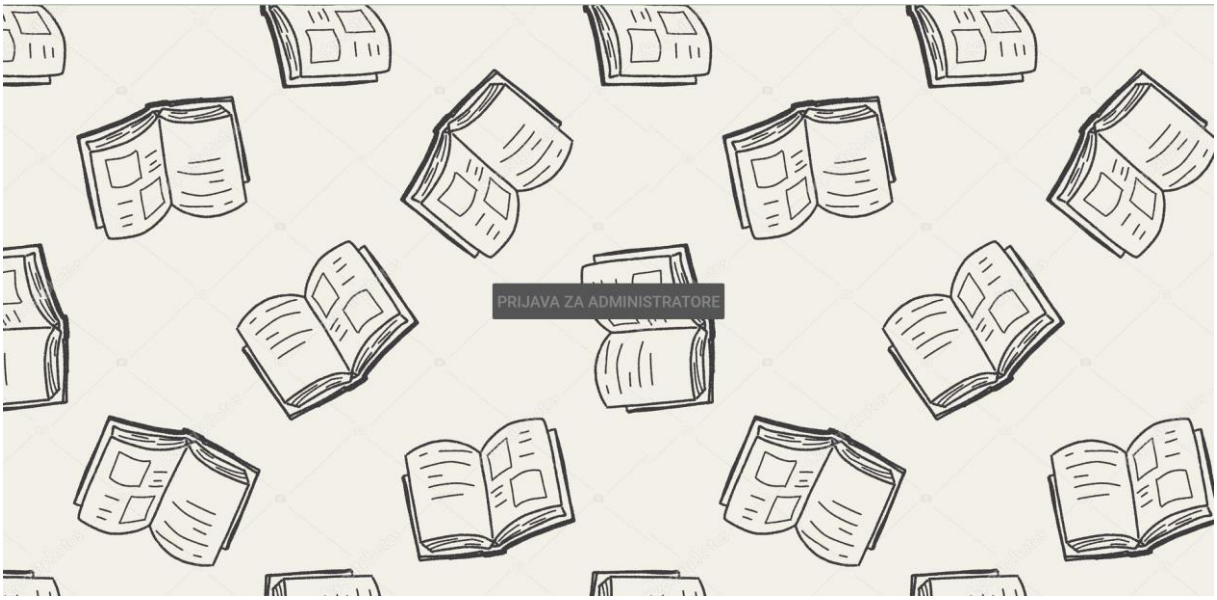
Poslužitelj (engl. Server) je računalo ili sustav koji putem računalne mreže pruža podatke, usluge ili programe drugim računalima mreže. Postoji mnogo vrsta poslužitelja, a neke od njih su: web poslužitelji, poslužiteljske pošte i virtualni poslužitelji [21]. Web poslužitelji su vrsta poslužitelja koji pohranjuju i pružaju sadržaj s web stranice (tekst, slike, video i sl.) na zahtjev klijenata. Uloge poslužitelja su: čekanje zahtjeva za dohvaćanje dokumenata koji se nalaze na poslužiteljskom računalu, dohvaćanje zatraženih dokumenata te njihova ispostava klijentima. Web poslužitelj komunicira s web preglednikom pomoću protokola za prijenos hiperteksta (engl. HyperText Transfer Protocol – HTTP), odnosno HTTPS (engl. HyperText Transfer Protocol Secure) protokola koji radi SSL/TLS enkripciju sadržaja kako bi se spriječile preglede sadržaja osobama koje bi mogli prisluškivati komunikaciju. Iz tog razloga, web poslužitelje također nazivamo i HTTP poslužitelje zbog korištenja HTTP protokola za komunikaciju.
S druge strane, postoji i razvojni model bez poslužitelja (engl. Serverless). Poslužitelj usluga bez poslužitelja nudi korisnicima pianje koda i njegovu implementaciju bez brige o temeljnoj strukturi (instalacija i postavljanje odgovarajućih programa). Iako se nazivaju „bez poslužitelja“, koriste se gotovi poslužitelji te ih tvrtke nude kao uslugu. Prednosti ove arhitekture su niži troškovi jer korisnici plaćaju samo za prostor koji su zauzeli na serveru i njegov izračun; ne moraju plaćati fiksne iznose za neiskorišten prostor ili vrijeme CPU-a u mirovanju. Također, smanjeno je i vrijeme koje bi inače bilo utrošeno u postavljanje i instalaciju odgovarajućih programa.
U web aplikaciji za stvaranje i upravljanje radnim nalozima korišten je Firebase Hosting kao primjer arhitekture bez poslužitelja [22]. Firebase Hosting omogućava postavljanje više vrsta web aplikacija a neke od njih su progresivne (PWA) i aplikacije s jednom stranicom (SPA). Također omogućuje uparivanje s Firebase Cloud funkcijama što je pomoglo u izradi web aplikacije za radne naloge. Posluživanje sadržaja putem sigurne veze, brzo dostavljanje sadržaja, podržavanje velikog broja vrsta sadržaja (HTML/CSS i drugih datoteka); sve su to razlozi za korištenje Firebase Hostinga. Za njegovo korištenje potrebni su nam Google račun te Firebase Cli.

5. Opis elemenata i funkcionalnost web aplikacije

Kao i druge aplikacije, web aplikacija za stvaranje i upravljanje radnim nalozima sastoji se od niza elemenata koji ju tvore. Svaki element ima svoju funkcionalnost u aplikaciji. U ovom poglavlju elementi aplikacije bit će opisati redoslijedom kako nastupaju u korisničkom sučelju.

5.1 Početna stranica

Prva stranica koja se prikaže korisniku kad pokrene web aplikaciju jest Početna stranica. Ona je zamišljena da bude što jednostavnija i funkcionalnija. Sadrži jedan gumb „Prijava za administratore“ koji preusmjerava korisnika na stranicu za prijavu te pozadinu koja tvori uzorak slika. Gumb je centriran u sredini stranice kako bi bio uočljiv i pregledan kao što je prikazano na Slici 1. Prilikom kodiranja izgleda i funkcionalnosti stranice korišteni su programski jezici: HTML/CSS i JavaScript. HTML kôd nalazi se pod oznakom *<template>* te deklarira klasu *pocetna* u kojoj se nalazi gumb; on opisuje ime i veličinu te poziva metodu *openLoginPage* koja se izvodi kada korisnik klikne na njega (*@click*). JavaScript kôd nalazi se pod oznakom *<script>* te je zadužen za metodu *OpenLoginPage* koja preusmjerava korisnika na stranicu *Login.* CSS kôd nalazi se pod oznakom *<style>* te opisuje izgled i stil varijabla koji su se deklarirali u HTML dijelu kôda; postavlja izgled klase *pocetna* i gumba *q-btn.*



Slika 1: Početna stranica

Kôd sadržan u datoteci PocetnaLayout.vue:

```
<template>
    <div class="pocetna">
      <q-btn
        flat
        dense
```

```
        @click="openLoginPage"
        label="Prijava za administratore"
        rounded="xl"
        x-large
        size=25px
        style="background-color:#555;color:#888;"
      />
    </div>
</template>

<script>
export default {
  name: 'PocetnaLayout',
  methods: {
    openLoginPage () {
      this.$router.push('/Login')
    }
  }
}
</script>

<style scoped>
  .pocetna {
    position: absolute;
    height: 100%;
    width: 100%;
    display: flex;
    align-items: center;
    justify-content: center;
    background-
image: url('https://st2.depositphotos.com/4393299/7373/v/950/depositphotos_73732407
-stock-illustration-doodle-book-seamless-pattern-background.jpg');
  }
</style>
```

Kako bi se stranica mogla prikazati treba joj se dodati putanja. U datoteci routes.js stvaraju se i pišu veze kako bi se stranica mogla prikazati na web pregledniku. Početnoj stranici pridodana je početna putanja *"/"* koja sadrži komponentu ili datoteku na koju se spaja veza, a to je *PocetnaLayout.vue.*

routes.js:

```
{
    path: '/',
    component: () => import('layouts/PocetnaLayout.vue')
},
```

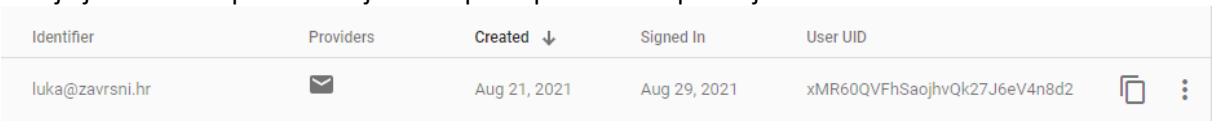
U slučaju da korisnik napiše i odabere putanju koja ne postoji, učitat će se *Error404.vue* komponenta. To je stranica koja obaviještava korisnika da je došlo do pogreške te da se vrati na početnu stranicu.

```
if (process.env.MODE !== 'ssr') {

  routes.push({
    path: '*',
    component: () => import('pages/Error404.vue')
  })
```

5.2 Autentifikacija

Nakon pritiska na gumb *Prijava za administratore,* korisnik se preusmjeri na stranicu za autentifikaciju korisnika - prijavu. Za postavljanje autentifikacije koristi se Firebase konzola, izbornik Authentication, koji omogućuje prijavu na različite načine: pomoću e-maila i lozinke, Google računa, Facebook računa i slično. Omogućena je prijava pomoću korisničkog imena i lozinke preko kojih se dodaje korisnik za

prijavu u web aplikaciju. Kao što je vidljivo na Slici 2, stvoren je korisnik *luka@zavrsni.hr* te mu je dodijeljena lozinka pomoću koje može pristupiti aplikaciji.



Slika 2: Primjer autentifikacije

Kako bi aplikacija mogla omogućiti prijavu korisnika dodijeljenog na Firebase konzoli, potrebno je povezati Firebase projekt i web aplikaciju. Pod *Project Overview* nalaze se postavke projekta *Project Settings.* Navedeni kôd potrebno je dodati u web aplikaciju, u datoteku *RadniNalogSetup.js.*

Kôd sadržan u datoteci RadniNalogSetup.js:

```
import firebase from 'firebase/app'

import 'firebase/auth'
import 'firebase/firestore'
import 'firebase/storage'
const firebaseConfig = {   (*)
  apiKey: 'AIzaSyC2ouXFBEdjjlkT52c9ttAI_xMEZ6e4x94',
  authDomain: 'radninalog---zavrsni-rad.firebaseapp.com',
  databaseURL: 'https://radninalog---zavrsni-rad-default-rtdb.europe-
west1.firebasedatabase.app',
  projectId: 'radninalog---zavrsni-rad',
  storageBucket: 'radninalog---zavrsni-rad.appspot.com',
  messagingSenderId: '11982759518',
  appId: '1:11982759518:web:fa29b8cad8bbf19e5b76cf',
  measurementId: 'G-SZQX8C6JKW'
}
firebase.initializeApp(firebaseConfig)
export default ({ Vue }) => {
  Vue.prototype.$auth = firebase.auth()   (**)
  Vue.prototype.$db = firebase.firestore()   (***)
  Vue.prototype.$storage = firebase.storage()   (****)
}
```

RadniNalogSetup.js sadrži informacije/konfiguracije(*) o Firebase projektu koje su potrebne kako bi web aplikacija i projekt mogle izmjenjivati podatke i međusobno komunicirati. Aplikacija se povezuje s Firebase bazom podataka(**), sustavom za autentifikaciju(***) i sustavom za postavljanje datoteka(****). Također, kako bi s izvela ova datoteka, potrebno ju je prijaviti u *quasar.conf.js* datoteku gdje se dodaje niz *boot*: [‘RadniNalogSetup.js’].
U datoteci *index.js* prijavljuju se putanje definirane u *routes.js.* Također, u njoj se radi i provjera korisničkih ovlasti nad nekom putanjom. Koristi se metoda *.beforeEach()* koja se izvodi pri svakom ulasku u neku putanju. Algoritam metode provjerava zahtjeva li putanja autorizirani pristup. Ukoliko da, provjera se obavlja nad korisnikom čime se poziva funkcija *isUserLoggedIn* koja vraća *Premise.* Nadalje, algoritam dopušta korisniku odlazak na željenu putanju ukoliko je on prijavljen; ako nije, vraća ga na početnu stranicu s putanjom /. Ako putanja ne zahtjeva autorizirani pristup, algoritam mu dopušta odlazak na željenu putanju.

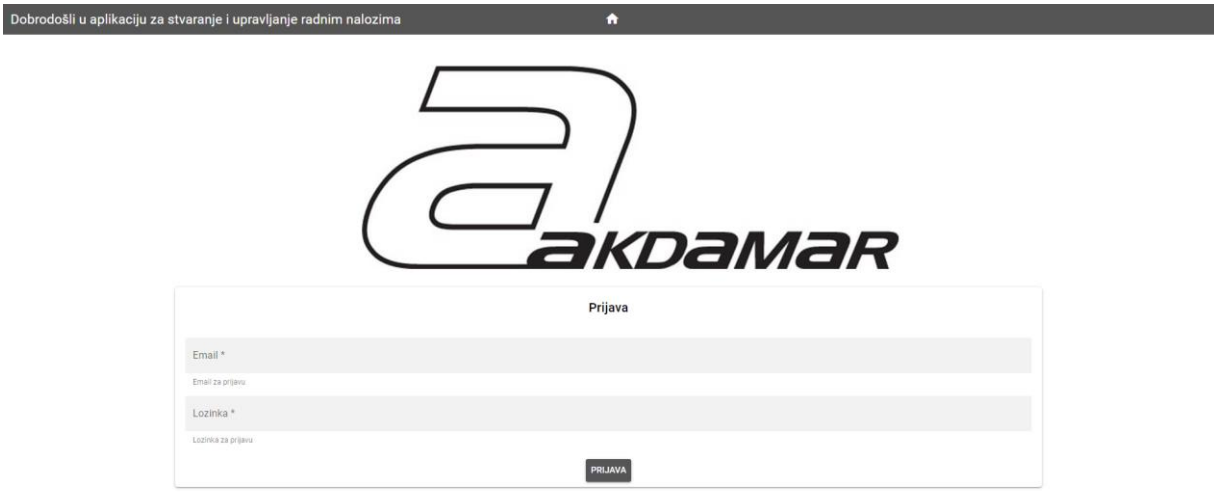
Kôd sadržan u datoteci index.js:

```
const isUserLoggedIn = () => {

  return new Promise((resolve, reject) => {
    const unsubscribeOnAuthStateChanged = Vue.prototype.$auth.onAuthStateChanged(th
eUser => {
      resolve(theUser)
      unsubscribeOnAuthStateChanged()
    }, err => {
      console.error(err)
```

```
      resolve(null)
      unsubscribeOnAuthStateChanged()
    })
  })
}
export default function (/* { store, ssrContext } */) {
  const Router = new VueRouter({
    scrollBehavior: () => ({ x: 0, y: 0 }),
    routes,
    // Leave these as they are and change in quasar.conf.js instead!
    // quasar.conf.js -> build -> vueRouterMode
    // quasar.conf.js -> build -> publicPath
    mode: process.env.VUE_ROUTER_MODE,
    base: process.env.VUE_ROUTER_BASE
  })
  Router.beforeEach(async (to, from, next) => {
    console.log(Vue.prototype.$auth.currentUser)
    if (to.matched.some(record => record.meta.auth)) {
      await isUserLoggedIn()
        .then(res => {
          if (res) {
            next()
          } else {
            next('/')
          }
        })
    } else {
      next()
    }
  })
  return Router
}
```

5.2.1 Prijava

Stranica za prijavu u zagljavlju *<q-header>* sadrži traku *<q-toolbar>* na kojoj se nalazi tekst dobrodošlice *<q-toolbar-tittle>* i ikona kuće koja funkcionira kao gumb *<q-btn>* koji prilikom klika *@click* preusmjerava natrag na početnu stranicu ('/') pomoću metode *OpenPocetnaPage*. Također, stranica sadrži sliku i sekciju za unos korisničkih podataka. Sekcija za unos korisničkih podataka sastoji se od naslova, polja za upis e-mail adrese, polja za upis lozinke i gumba *Prijava* kao što je vidljivo iz Slike 3.



Slika 3: Stranica za prijavu

Kôd sadržan u datoteci LoginLayout.vue:

```
<template>

 <q-layout>
 <q-header>
 <q-toolbar>
 <q-toolbar-title>
     Dobrodošli u aplikaciju za stvaranje i upravljanje radnim nalozima
 </q-toolbar-title>
 <q-btn
  flat
  dense
  @click="OpenPocetnaPage"
  icon="home"
  />
 <q-space />
 </q-toolbar>
 </q-header>
 <q-page-container>
   <div class="slika"/>
 <router-view />
 </q-page-container>
 </q-layout>
</template>
<script>
export default {
  name: 'LoginLayout',
  methods: {
    OpenPocetnaPage () {
      this.$router.push('/')
    }
  }
}
</script>
```

Sekcija *<q-card-section>* sadrži klase *<q-input>* za unos podataka o e-mail adresi i lozinci. Pri upisu e-maila postavljeno je pravilo pomoću *emailPattern* metode koje provjerava zadovoljava li unesena riječ e-mail uzorak. U *script* dijelu kôda definira se aplikativna logika. Sastoji se od dijelova *name* koja pridodaje komponenti jedistveni naziv, a to je *LoginIndex. Data* sadrži popis svih varijabli čija promjena izaziva promjenu u korisničkom sučelju. U kôdu se nalaze dvije varijable: email i password koje se koriste kao model u koji se spremaju podaci koji se unose pomoću komponente *<q-input>. Mounted* je standardna metoda koja se automatski poziva kada je neka komponenta spremna. Ona provjerava je li korisnik prijavljen u sustav pomoću naredbe *this.$auth.currentUser*. *$auth* je globalno dostupan objekt firebasea koji je definiran u datoteci *RadniNalogSetup.js*. Ako postoji vrijednost u atributu *currentUser*, odnosno ako je korisnik prijavljen, tada se njega preusmjerava na putanju */Izbornik*. *OnLogin* je metoda koja se poziva kada korisnik klikne na gumb *Prijava.* U njoj se poziva metoda *signInWithEmailAndPassword* objekta *$auth* kojoj se preko parametara *this.email* i *this.password* dostavljaju uneseni e-mail i lozinka. Metoda vraća objekt *Promise* koji predstavlja završetak ili grešku u pozivu metoda. *Promise* se sastoji od dva dijela: *.then* koja se izvodi kada je uspješno izvedena metoda, i *.catch.* koja se izvoda kada je neuspješno izvedena metoda. Kada se korisnik uspješno prijavi, tada se blok *.then* uspješno izvodi i preusmjerava korisnika na putanju */Izbornik.* U slučaju da je došlo do pogreške u prijavi, u *.catch* bloku se poziva metoda *notify* objekta *$q* koja prikazuje poruku korisniku o nespješnoj prijavi.

Kôd sadržan u datoteci LoginIndex.vue:

```
<template>

  <div class="row justify-center">
  <div class="col">
  <q-form @submit="onLogin">
  <q-card>
```

```
  <q-card-section align="center">
    <div class="text-h6 border">
      Prijava
    </div>
  </q-card-section>
  <q-card-section>
    <div class="q-gutter-md">
    <q-input
      filled
      type="email"
      v-model="email"
      label="Email *"
      hint="Email za prijavu"
      lazy-rules
      :rules="[ val => emailPattern.test(val) || 'Molimo napišite email']"/>
    <q-input
      filled
      type="password"
      v-model="password"
      label="Lozinka *"
      hint="Lozinka za prijavu"
      lazy-rules
      :rules="[ val => val && val.length > 0 || 'Molimo napišite lozinku']"/>
  </div>
 </q-card-section>
 <q-card-actions align="center">
  <q-btn
    label="Prijava"
    type="submit"
    color="rgb(202, 249, 123)"/>
 </q-card-actions>
 </q-card>
 </q-form>
 </div>
 </div>
</template>
<script>
export default {
  name: 'LoginIndex',
  data () {
    return {
      emailPattern: /^(?=[a-zA-Z0-9@._%+-]{6,254}$)[a-zA-Z0-9._%+-]{1,64}@(?:[a-zA-
Z0-9-]{1,63}\.){1,8}[a-zA-Z]{2,63}$/,
      email: null,
      password: null
    }
  },
  mounted: function () {
    if (this.$auth.currentUser) {
      this.$router.push('/Izbornik')
    }
  },
  methods: {
    onLogin () {
      this.$auth.signInWithEmailAndPassword(this.email, this.password)
        .then(response => {
          this.$router.push('/Izbornik')
        })
        .catch(error => {
          console.log(error)
          this.$q.notify({
            type: 'negative',
            message: 'Prijava neuspješna.'
          })
        })
    }
  }
}
```

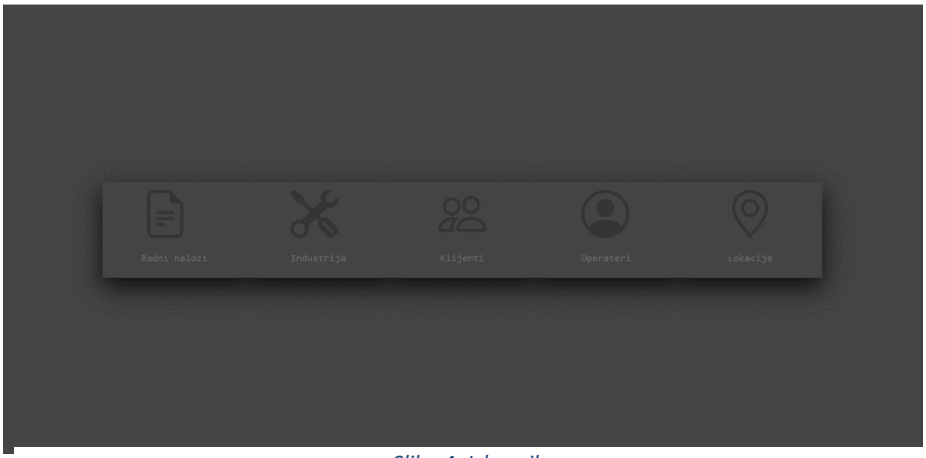
```
</script>
<style scoped>
  .q-btn {
    background-color: #555;
  }
  .col {
    height: 45%px;
    max-width: 71.6%;
    size: 1000px;
    align-content: center;
    justify-content: center;
    padding-left: 0%;
  }
  .slika {
    padding-left: 20%;
  }
  </style>
```

Stranici za prijavu postavljena je putanja */Login* koja sadrži komponentu ili datoteku s kojom se veže, a to je *LoginLayout.vue.* Također, postavljena je dodatna veza, tzv. dijete koje se veže na komponentu *LoginLayout.vue*, a to je *LoginIndex.vue.* Razlog podijele stranice za prijavu u dvije datoteke je taj što se u jednoj datoteci (*LoginLayout.vue*) stvara sučelje i komponente izgleda, dok se u drugoj (*LoginIndex.vue*) izrađuju funkcije; sve kako bi se imala veća preglednost nad kôdom.

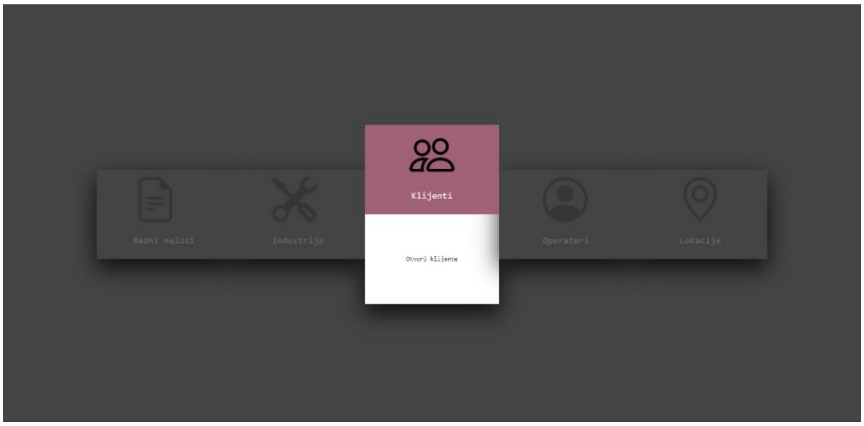
```
{

    path: '/Login',
    component: () => import('layouts/LoginLayout.vue'),
    children: [
      { path: '', component: () => import('pages/LoginIndex.vue') }
    ]
```

5.3 Izbornik

Nakon pritisnka na gumb *Prijava* te provjere autentifikacije, korisnika se preusmjerava na stranicu *Izbornik.* Stranica se temelji na jednostavnim elementima kako bi se korisniku olakšalo njezino korištenje. Sastoji se pet kartica koje se, kad se usmjeri pokazivač na svaku od njih, prikažu (prikazano na Slici 5). Svaka kartica sastoji se od naslova i ikone koje opisuju sadržaj i funkcionalnost koje nose te od gumba koji otvara novu stranicu sadržaja. Tako imamo kartice: Radni nalozi, Industrija, Klijenti, Operateri i Lokacije (prikazano na Slici 4).



Slika 4: Izbornik



Slika 5: Izbornik kada se pokazivač umjeri na karticu Klijenti

Kôd stranice *Izbornik* načinjen je većinom iz programskog jezika HTML/CSS. U klasi *container* stvoreno je pet kartica klase *card.* One sadržavaju preuzete ikone *<svg>* koje ih opisuju te naslov *h3* koji ih imenuju. Također, kartice sadržavaju komponente *<q-item>* koje prilikom klika (click), preusmjeravaju korisnik na zadanu putanju. CSS kôd zadužan je za boju, font i veličinu teksta i kartica na stranici, *hover* funkciju koja omogućuje da se prilikom usmjeravanja pokazivača na karticu ona otvori i pokaže sadržaj.

Kôd sadržan u datoteci IzbornikLayout.vue

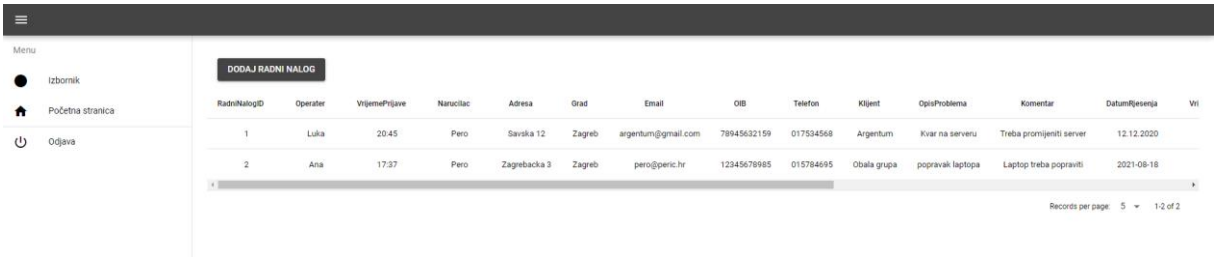
```
<div class="container">

        <div class="card">
            <div class="face face1">
                <div class="content">
                    <svg xmlns="http://www.w3.org/2000/svg" width="100" height="100
" fill="currentColor" class="bi bi-file-earmark-text" viewBox="0 0 16 16">
                      <path d="M5.5 7a.5.5 0 0 0 0 1h5a.5.5 0 0 0 0-1h-
5zM5 9.5a.5.5 0 0 1 .5-.5h5a.5.5 0 0 1 0 1h-5a.5.5 0 0 1-.5-.5zm0 2a.5.5 0 0 1 .5-
.5h2a.5.5 0 0 1 0 1h-2a.5.5 0 0 1-.5-.5z"/>
                      <path d="M9.5 0H4a2 2 0 0 0-
2 2v12a2 2 0 0 0 2 2h8a2 2 0 0 0 2-
2V4.5L9.5 0zm0 1v2A1.5 1.5 0 0 0 11 4.5h2V14a1 1 0 0 1-1 1H4a1 1 0 0 1-1-
1V2a1 1 0 0 1 1-1h5.5z"/>
                    </svg>
                    <h3>Radni nalozi</h3>
                </div>
            </div>
            <div class="face face2">
                <q-item
                    clickable
                    to="/Podaci"
                >
                <q-item-section>
                    <q-item-label>Otvori radne naloge</q-item-label>
                </q-item-section>
                </q-item>
            </div>
        </div>
```

Za prikazivanje i povezivanje putanje stranice *Izbornik,* u datoteci *routes.js* dodana je putanja */Izbornik* na datoteku *IzbornikLayout.vue.* Stranici se može pristupiti ukoliko je objekt *$auth* istinit, tj ako je autentifikacija uspješno provedena.

```
{

    path: '/Izbornik',
    component: () => import('layouts/IzbornikLayout.vue'),
    meta: { auth: true }
},
```

5.4 Sučelje sustava

Kada korisnik pritisne na gumb *Otvorit radne naloge,* preusmjerava ga se na putanju */RadniNalog*. Stranica *RadniNalog* sadrži informacije o radnim nalozima te omogućava njegove upise. Pošto je putanja stranice komponenta glavne komponente */Admin,* stranica se učitava u sučelje kreirano u datoteci *SystemLayout.vue.* U datoteci *SystemLayout.vue* definirana je traka u zaglavlju stranice i gumb koji služi kao izbornik i sadrži komponente: *Izbornik* pomoću koje se korisnik vraća na Izbornik stranicu; *Početna stranica* pomoću koje se korisnik vraća na Početnu stranicu i *Odjava* (prikazano na Slici 6) koja služi kako bi se korisnik mogao odjaviti iz administrativnog sučelja web aplikacije.



Slika 6: Sučelje sistema i stranice RadniNalog

U kôdu datoteke *SystemLayout.vue* nalaze se oznake koje opisuju svaki element sučelja. Tako imamo komponentu *q-btn* koja predstavlja gumb prikazan na zaglavlju stranice. Kada korisnik klikne na njega aktivirat će se događaj *click* unutar kojeg se postavlja varijabla *leftDrawerOpen.* Komponenta *q-drawer* predstavlja bočnu traku koja se otvara/zatvara ovisno o varijabli *leftDrawerOpen* i sadrži druge komponente. Unutar komponente *q-list* grupiraju se komponente *q-item*, *q-item-section* i *q-item-label.* Komponenta *q-item* zadužena je za pozivanje putanji ukoliko korisnik klikne na nju. Unutar komponente *q-item-section* postavlja se ikona i naziv menija u isti redak. *Q-item-label* koristi se za prikazivanje naslova menija, što je u ovom slučaju korišteno kao prazan redak radi veće preglednosti ostalih komponenti menija. *Q-icon* komponenta prikazuje ikone. Komponenta *q-page-container* prikazivat će druge komponente, a *<router-view />* komponenti se postavlja druga komponenta na koju se korisnik usmjerava, a to je */Nalog*.

U skriptnom dijelu datoteke nalaze se dijelovi koji definiraju aplikativnu logiku. Dio koji označava jedinstven naziv komponente naziva se *name*, a to je naziv datoteke u kojoj se nalazi korisnik *SystemLayout. Data* sadrži popis svih varijabli čija promjena izaziva promjenu u korisničkom sučelju, a u ovom slučaju to je varijabla *leftDrawerOpen.* Prilikom klika na komponentu *Odjava* iz menija*,* metoda *logout* se poziva i izvodi firebase metodu *signOut()* pomoću koje se korisnik uspješno odjavljuje iz administrativnog sučelja aplikacije te se preusmjeri na stranicu *Login*. Metoda *home* nakon što korisnik klikne na komponentu *Početna stranica* se pozove te ona preusmjeri korisnika na početnu stranicu aplikacije.

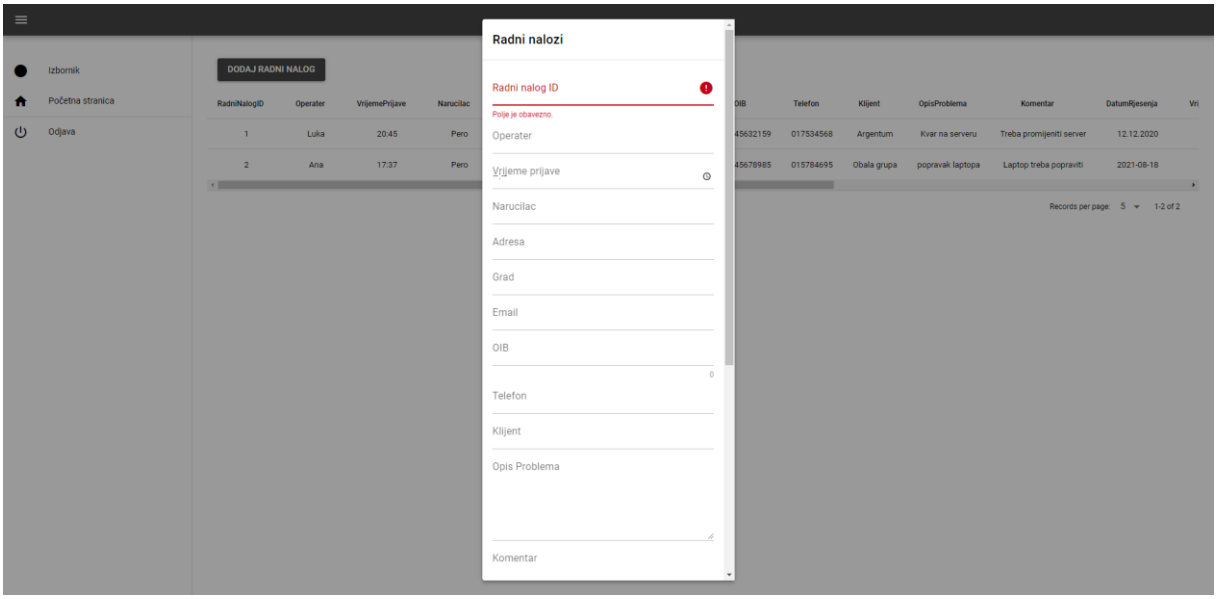
Dio kôda sadržan u datoteci SystemLayout.vue:

```
<q-drawer
 v-model="leftDrawerOpen"
 show-if-above
 bordered
 >
 <q-list>
 <q-item-label header></q-item-label>
 <q-item
 Clickable
 to="/Izbornik"
>
 <q-item-section avatar>
 <q-icon name="circle" />
 </q-item-section>
 <q-item-section>
 <q-item-label>Izbornik</q-item-label>
 </q-item-section>
 </q-item>
 <q-item
 clickable
 @click="home"
 >
 <q-item-section avatar>
 <q-icon name="home" />
 </q-item-section>
 <q-item-section>
```

```
<script>
export default {
  name: 'SystemLayout',
  data () {
    leftDrawerOpen: false
  },
  methods: {
    logout () {
      .then(() => {
          this.$router.push('/Login')
       })
     },
     home () {
       this.$router.push('/')
     }
   }
  }
  </script>
```

5.4.1 Radni nalozi

Nakon što nam je učitano sučelje sustava, učitava se i stranica */RadniNalozi*. Pošto su *RadniNalozi* dio sučelja *SystemLayot,* stvoreni su kao sadržaj *q-page-containera.* Stranica se sastoji od gumba *Dodaj radni nalog* i podataka koji su prikazani u organiziranoj tablici. Na kraju tablice, nalaze se dva gumba za izmjenu i brisanje retka tablice – *Uredi* i *Obriši.* Prilikom pritiska na gumb *Dodaj radni nalog* prikazuje se novi prozor (dijalog) za unos podataka koji će nakon unosa biti prikazani u tablici (Slika 7).



Slika 7: Stranica RadniNalog

U *template* dijelu datoteke definirano je korisničko sučelje. Kôd će se objašnjavati redoslijedom kako se definiraju blokovi i komponente. U njemu se nalaze komponente i blokovi koji definiraju stranicu. Osim blokova div u kojima će se smjestiti svi elementi sučelja, korištene su komponenta *q-form* koja predstavlja formu u koju korisnik unosi podatke i *q-table* komponenta koja prikazuje unesene podatke tablično.
Također, u komponenti *q-table* koristi se atribut *v-if* kako bi se specificirao uvjet koji treba biti zadovoljen kako bi se komponenta prikazala, što je u ovom slučaju postavljen na *isMounted*. Atribut *title*, imenovan kao *Nalozi*, koristi se radi nazivanja naslova tablice. Atribut *row-key* korišten je za jedinstveno identificiranje retka iz polja liste, što najčešće čini primarni ključ. U kodu je naveden *UIDnalozi* jer predstavlja jedistvenu identifikaciju dokumenta. Atribut *flat* predstavlja ravan izgled tablice, bez iscrtanih granica.
Komponenta *<template v-slot:A=B>* postavlja komponente djece unutar definiranog slota komponente roditelj. Drugim riječima, umjesto slova A unutar naziva komponente, postavlja se naziv slota, a umjesto slova B postavlja se naziv varijable. Naziv varijable postavlja se samo u slučaju ako su u slotu A navedeni nekakvi podaci. Komponenta *<template v-slot:top>* postavlja slot *top* preko kojega se druge komponente mogu postaviti na vrh stranice. Pomoću tog slota, postavlja se gumb *Dodaj radni nalog* na vrh stranice te klikom na njega poziva se metoda *onNewRow.* Komponenta *<template v-slot:body-cell="props">* posjeduje slot *body-cell* pomoću kojeg se definiraju ćelije tablice. Nadalje, za njihovo definiranje korištena je komponenta *<q-td>* koja pomoću atributa :props dostavlja komponentama djeci podatke o kolonama i retcima tablice. Definirani su gumbi za brisanje i izmjenu podataka koji se pojavljuju u koloni *actions - .* Kad korisnik klikne na gumb za brisanje, poziva se metoda *onDeleteRow*. Ukoliko korisnik pritisne drugi gumb, za izmjenu, poziva se metoda *onUpdateRow.*
Komponenta *q-dialog* služi kako bi se otvori dijalog. U sebi sadrži atribute *v-if* i *v-model* koji se koriste kako bi se navelo treba li ili ne treba otvoriti dijalog. Korištena je varijabla *openDialog* koju pomoću vrijednosti *true* i *false* navodimo na otvaranje, tj. zatvaranje dijaloga.

Komponenta *q-card* prikazuje druge komponente u obliku grupirane kartice te je moguće unutar nje postavljati nove komponente. *Q-card-section* je komponenta koja se koristi kako bi se prikazao blok unutar kartice. Ona omogućuje prikaz komponenti koji korisnik unosi preko komponente *q-input.* Pomoću Komponente *q-input*, korisnik unosi podatak. Koristi se više komponenti za unos, a to su: RadniNalogID, Operater, VrijemePrijave, Naručilac, Adresa, Grad, Email, OIB, Telefon, Klijent, OpisProblema, Komentar, DarumRješenja, VrijemeRješenja, UtrošenoRadnihSati, Fakturirati, PokrivenoUgovorom, Materijal, RacunBroj, Status. Ove *q-input* komponente imaju postavljenu jedistvenu identifikaciju komponente, zvanu *ref*, pomoću koje se provjerava je li validacija podatka prošla uspješno. Atributom *label* definirane su oznake polja, primjerice *label="RadniNalogID"*. U prvoj komponenti *q-input, RadniNalogID,* definirano je pravilo ispravnosti podataka (*:error)*, koje ispisuje poruku „Polje je obavezno.“ ukoliko korisnik ne upiše nikakvu vrijednosti u to polje. Objekt *radninalog* koristi se za spremanje unesene komponente; pridodaje mu se atribut, primjerice *RadniNalogID* na način *radninalog.RadniNalogID* gdje se objekt *radninalog* sastoji od atributa *RadniNalogID.* Komponenta *q-card-actions* poseban je blok unutar kartice koja sadrži dvije –btn varijable: *U redu* i *Odustani.* Pritiskom na *U redu* gumb poziva se metoda *onOKClick,* dok se pristiskom na *Odustani* gumb poziva *onCancelClick* metoda.

Dio kôda sadržan u datoteci RadniNalogIndex.vue

```
<q-dialog

      v-if="openDialog"
      v-model="openDialog"
    >
      <q-card class="q-dialog-plugin">
        <q-card-section>
          <div class="text-h6">
```

```
            Radni nalozi
          </div>
        </q-card-section>
        <q-separator />
        <q-card-section>
          <q-input
            ref="RadniNalogID"
            type="number"
            :error="!radninalog.RadniNalogID || radninalog.RadniNalogID.length ===
0"
            error-message="Polje je obavezno."
            label="Radni nalog ID"
            v-model="radninalog.RadniNalogID"
          />
          <q-input
            ref="Operater"
            type="text" multiple options
            label="Operater"
            v-model="radninalog.Operater"
          />
```

U *script* dijelu koda definira se aplikativna logika. Sastoji se od dijelova poput *name*, koji predstavlja jedinstveni naziv komponente, a to je *RadniNalogIndex*.
U *data* dijelu (popis varijabli čija primjena izaziva promjenu na sučelju) nalaze se sljedeće varijable: *openDialog*, *radninalog, radninalogModel, isMounted, nalozi* i *columns.* Varijabla *openDialog* specificirana je za otvaranje i zatvaranje dijaloga za unos i izmjenu podataka. Varijabla *radninalog* koriste se za izmjenu podataka u dijalogu. Varijabla u kojoj se nalazi struktura podataka koja se nakon toga sprema u bazu podataka kao novi dokument naziva se *radninalogModel*. Ona se sastoji od sljedećih atributa: RadniNalogID, Operater, VrijemePrijave, Naručilac, Adresa, Grad, Email, OIB, Telefon, Klijent, OpisProblema, Komentar, DarumRješenja, VrijemeRješenja, UtrošenoRadnihSati, Fakturirati, PokrivenoUgovorom, Materijal, RacunBroj, Status. Varijabla koja označava završno stanje komponente naziva se *isMounted.* Varijabla *nalozi* označava listu svih podataka iz baze podataka. Varijabla *columns* označava listu kolona koje će se prikazati u tablici te se sastoji od objekta koje sadrže atribute poput *name, label, align, field* i *sortable.* Atribut *name* predstavlja jedistvenu identifikaciju stupca, *align* služi za poravnanje ćelije, *label* označava naslov stupca, *field* naziv polja u bazi podataka, dok *sortable* služi za sortiranje podataka po stupcu. Također, definiran je i stupac *actions* u kojem se nalaze gumbi *Uredi* i *Obriši* koji služe za brisanje ili naknadno uređivanje retka tablice. Kada se pozove komponenta *mounted*, stvara se referenca na kolekciju *radninalog* u firestore bazi podataka preko koje se dolazi pomoću javnog dostupnog atributa *this.$db* definiranog u datoteci *RadniNalogSetup.vue.* Pomoću firestore metode *get*, koja nam vraća sve retke, tj. dokumente iz kolekcije *radninalog,* referenca se sprema u varijablu *colectionRef*. Koristeći firestore metodu *push* prebacuje sve vraćene retke u varijablu *nalozi.* Nakon toga, pomoću metode *data()*, u listu *nalozi* ubacujemo podatke jednog retka. Kada se proces završio, varijabla *isMounted* postavlja se na *true* kako bi se podaci i stupci prikazali u tablici.
Pritiskom na gumb *Dodaj radni nalog*, poziva se metoda *onNewRow.* Pomoću metode se u varijablu *radninalog* stvara novi objekt strukture *radninalogModel*. Nakon toga, varijabli *openDialog* se pridodaje vrijednost *true* čime se postiže prikaz dijaloga sa varijablom *nalozi.* Pritiskom na gumb *U redu* u dijalogu, poziva se metoda *onOKClick*. Metoda provjera jesu li podaci uspješno prošli validaciju specifirana na *q-input* komponenti. Ako nije došlo do pogreške u komponentama, referenca se sprema u varijablu *collectionRef* prema kolekciji *radninalog.* Nakon toga, radi se provjera je li došlo do novog retka. Ukoliko je, tada se pomoću firestore metode za dodavanje novih dokumenata dodaje novi dokument u kolekciju *radninalog.* Prilikom uspješnog dodavanja, u atribut *radninalog.UIDradninalog* se sprema identifikacija dokumenta koju Firestore automatski postavlja. Referenca se dodaje prema novom spremljenom dokumentu kako bi se u njemu ažurirao atribut *UIDradninalog* pomoću metode *update.* Ukoliko je ažuriranje uspješno obavljeno, dodaje se radni nalog u listu radnih naloga čime se ažurira i sama tablica. Varijabla *openDialog* postavlja se na

false kako bi se zatvorio dijalog prilikom uspješnog izvršenja svih naredbi. Ako je došlo do neke greške prilikom ažuriranja ili dodavanja dokumenta, ispisuje se poruka „Pogreška pri dodavanju dokumenta“. Ukoliko nije došlo do novog reda, referenca se dohvaća prema dokumentu. Pomoću firestore metode dokumenta *set,* stari dokument zamjenjuje se novim. Prilikom uspješnog postavljanja, radni nalog poslan u bazu podataka pronalazi se u listi radnih nalog te se ažurira. Varijabla *openDialog* postavlja se na *false* kako bi se zatvorio dijalog, a ukoliko je došlo do pogreške u izvođenju kôda, ispisuje se poruka pogreške.
Prisikom na gumb *Odustani* u dijalogu, poziva se metoda *onCancelRow* koja postavlja varijablu *openDialog* na *false* te se time dijalog zatvara.
Pritiskom na gumb za izmjenu retka *Uredi*, poziva se metoda *onUpdateRow*. On funkcionira na način da se u varijablu *radninalog* dodaje novi objekt strukture *radninalogModel* te u njega prepisuje podatke retka dobiveni kroz argument metode. Nakon dodavanja novog objekta, varijabla *openDialog* postavlja se na *true* čime se otvara dijalog.
Pritiskom na gumb za brisanje retka *Obriši,* poziva se metoda *onDeleteRow.* Nakon pritiska, otvara se dijalog koji traži potvrdu brisanja. On sadrži poruku „Potvrdi brisanje“ i dva gumba: *U redu* i *Odustani.* Ukoliko korisnik pritisne na gumb *U redu,* poziva se metoda *onOK* koja dohvaća referencu na dokument. On se pomoću firestore metode za brisanje dokumenta *delete* briše. Ukoliko je brisanje uspjelo, pronalazi se indeks obrisanog radnog nalog u listi te se uklanja iz liste pomoću metode *splice.* U slučaju pogreške pri brisanju ispisuje se poruka pogreške „Pogreška pri brisanju dokumenta“.

Dio kôda sadržan u datoteci RadniNalogIndex.vue

```
<script>

export default {
  name: 'RadniNalogIndex',
  data () {
    return {
      openDialog: false,
      radninalog: null,
      radninalogModel: {
        UIDnalozi: null,
        RadniNalogID: null,
        Operater: null,
        VrijemePrijave: null,
        Narucilac: null,
        Adresa: null,
        Grad: null,
        Email: null,
        OIB: null,
        Telefon: null,
        Klijent: null,
        OpisProblema: null,
        Komentar: null,
        Datumrjesenja: null,
        VrijemeRjesenja: null,
        UtrosenoRadnihSati: null,
        Fakturirati: null,
        PokrivenoUgovor: null,
        Materijal: null,
        RacunBroj: null,
        Status: null
      },
      isMounted: false,
      nalozi: [],
      columns: [
        {
          name: 'RadniNalogID',
          label: 'RadniNalogID',
          align: 'center',
          field: 'RadniNalogID',
          sortable: true
        },
```

```
        …
  mounted: function () {

    const collectionRef = this.$db.collection('radninalog')
    collectionRef.get()
      .then((rows) => {
        rows.forEach((row) => {
          this.nalozi.push(row.data())
        })
        this.isMounted = true
      })
  },
  methods: {
    onNewRow () {
      this.radninalog = JSON.parse(JSON.stringify(this.radninalogModel))
      this.openDialog = true
    },
    onOKClick () {
      if (!this.$refs.RadniNalogID.hasError &&
      !this.$refs.hasError) {
        const collectionRef = this.$db.collection('radninalog')
        if (this.radninalog.UIDnalozi === null) {
          collectionRef.add(this.radninalog)
            .then((doc) => {
              this.radninalog.UIDnalozi = doc.id
              const docRef = this.$db.collection('radninalog').doc(doc.id)
              docRef.update({ UIDnalozi: doc.id })
                .then((response) => {
                  this.radninalog.UIDnalozi = doc.id
                  this.nalozi.push(this.radninalog)
                  this.openDialog = false
                })
                .catch(function (error) {
                  console.error('Pogreška pri dodavanju dokumenta: ', error)
                })
            })
            .catch(function (error) {
              console.error('Pogreška pri dodavanju dokumenta: ', error)
            })
        } else {
          const docRef = this.$db.collection('radninalog').doc(this.radninalog.UIDn
alozi)
          docRef.set(this.radninalog)
            .then((response) => {
              const radninalog = this.nalozi.find(radninalog => radninalog.UIDnaloz
i === this.radninalog.UIDnalozi)
              if (radninalog) {
                for (const attributeName in this.radninalog) {
                  radninalog[attributeName] = JSON.parse(JSON.stringify(this.radnin
alog[attributeName]))
                }
              }
              this.openDialog = false
            })
            .catch(function (error) {
              console.error('Pogreška pri dodavanju dokumenta: ', error)
            })
        }
      }
    },
    onCancelClick () {
      this.openDialog = false
    },
    onUpdateRow (radninalog) {
      this.radninalog = JSON.parse(JSON.stringify(this.radninalogModel))
      for (const attributeName in this.radninalog) {
        this.radninalog[attributeName] = JSON.parse(JSON.stringify(radninalog[attri
buteName]))
```

```
      }
      this.openDialog = true
    },
    onDeleteRow (row) {
      this.$q.dialog({
        title: 'Obriši',
        message: 'Potvrdi brisanje.',
        ok: true,
        cancel: true
      }).onOk(() => {
        const docRef = this.$db.collection('radninalog').doc(row.UIDnalozi)
        docRef.delete()
          .then(() => {
            const index = this.nalozi.findIndex(radninalog => radninalog.UIDnalozi
=== row.UIDnalozi)
            if (index >= 0) {
              this.nalozi.splice(index, 1)
            }
          }).catch((error) => {
            console.error('Pogreška pri brisanju dokumenta: ', error)
          })
      })
    }
  }
}
</script>
```

Za prikazivanje i povezivanje putanje stranice *RadniNalog,* u datoteci *routes.js* dodana je putanja */RadniNalog* na datoteku *RadniNalogIndex.vue* . Stranici se može pristupiti ukoliko je objekt *$auth* istinit, tj. ako je autentifikacija uspješno provedena.

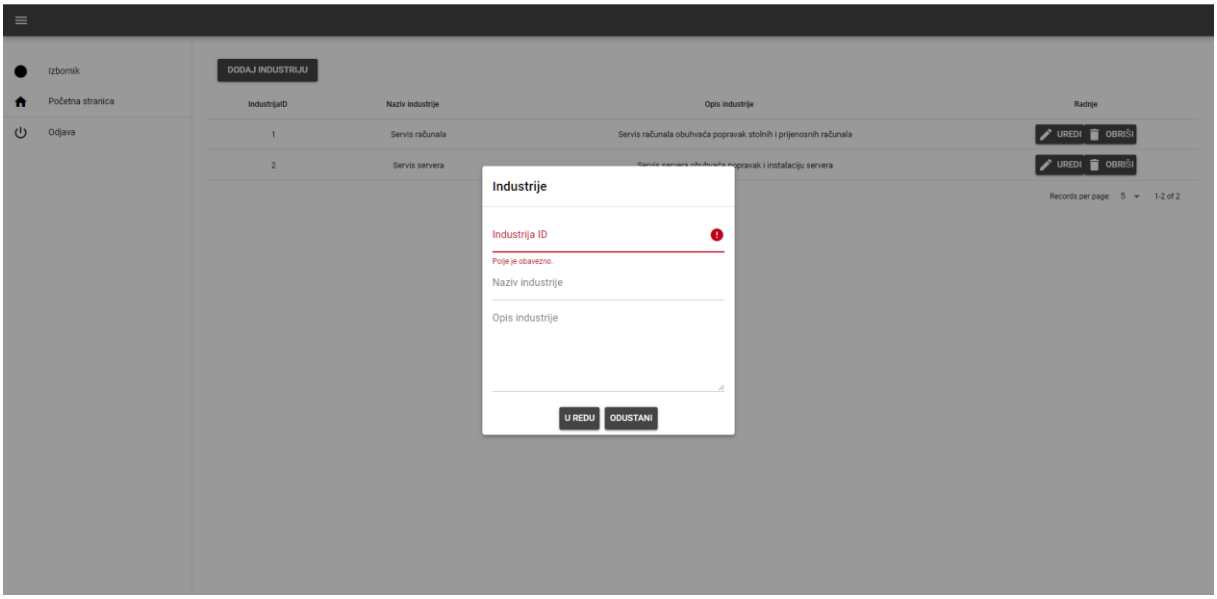
routes.js – dio koda:

```
        path: '/Admin',

    component: () => import('layouts/SystemLayout.vue'),
    meta: { auth: true },
    children: [
      { path: '/RadniNalog', meta: { auth: true }, component: () => import('pages/R
adniNalog/RadniNalogIndex.vue') },
```

5.4.2 Industrija

Kada korisnik u *Izborniku* pritisne na gumb *Otvori industriju,* preusmjerava se na putanju */Industrija.* Stranica se sastoji od gumba *Dodaj industriju* i podataka koji su prikazani u organiziranoj tablici. Na kraju tablice, nalaze se dva gumba za izmjenu i brisanje retka tablice – *Uredi* i *Obriši.* Pri pritiskom na gumb *Dodaj industriju* prikazuje se novi prozor(dijalog) za unos podataka koji će nakon unosa biti

prikazani u tablici (Slika 8).



Slika 8: Stranica Industrija

U *template* dijelu datoteke definirano je korisničko sučelje. Ono je definirano s istim varijablama i metodama kao i kod stranice *RadniNalozi.*
Komponenta *q-input* pomoću koje korisnik unosi podatak. Koriste se više komponenti za unos, a to su: IndustrijaID, Naziv industrije i Opis industrije. Ove *q-input* komponente imaju postavljenu jedistvenu identifikaciju komponente, zvanu *ref*, pomoću koje se provjerava je li validacija podatka prošla uspješno. Atributom *label* definirane su oznake polja, primjerice *label="IndustrijaID"*. U prvoj komponenti *q-input, IndustrijaID,* definirano je pravilo ispravnosti podataka (*:error)*, koje ispisuje poruku „Polje je obavezno." ukoliko korisnik ne upiše nikakvu vrijednosti u to polje. Objekt *industrija* koristi se za spremanje unesene komponente; pridodaje mu se atribut, primjerice *IndustrijaID* na način *industrija.IndustrijaID* gdje se objekt *industrija* sastoji od atributa *IndustrijaID.* Komponenta *q-card-actions* poseban je blok unutar kartice koja sadrži dvije –btn varijable: *U redu* i *Odustani.* Pritiskom na *U redu* gumb poziva se metoda *onOKClick,* dok se pristiskom na *Odustani* gumb poziva *onCancelClick* metoda.

Dio template kôda sadržan u datoteci IndustrijaIndex.vue

```
    <q-dialog
      v-if="openDialog"
      v-model="openDialog">
      <q-card class="q-dialog-plugin">
        <q-card-section>
          <div class="text-h6">
              Industrije
          </div>
        </q-card-section>
        <q-separator />
        <q-card-section>
          <q-input
            ref="IndustrijaID"
            type="number"
            :error="!industrija.IndustrijaID || industrija.IndustrijaID.length ===
0"
            error-message="Polje je obavezno."
            label="Industrija ID"
            v-model="industrija.IndustrijaID"/>
          <q-input
            ref="Naziv"
```

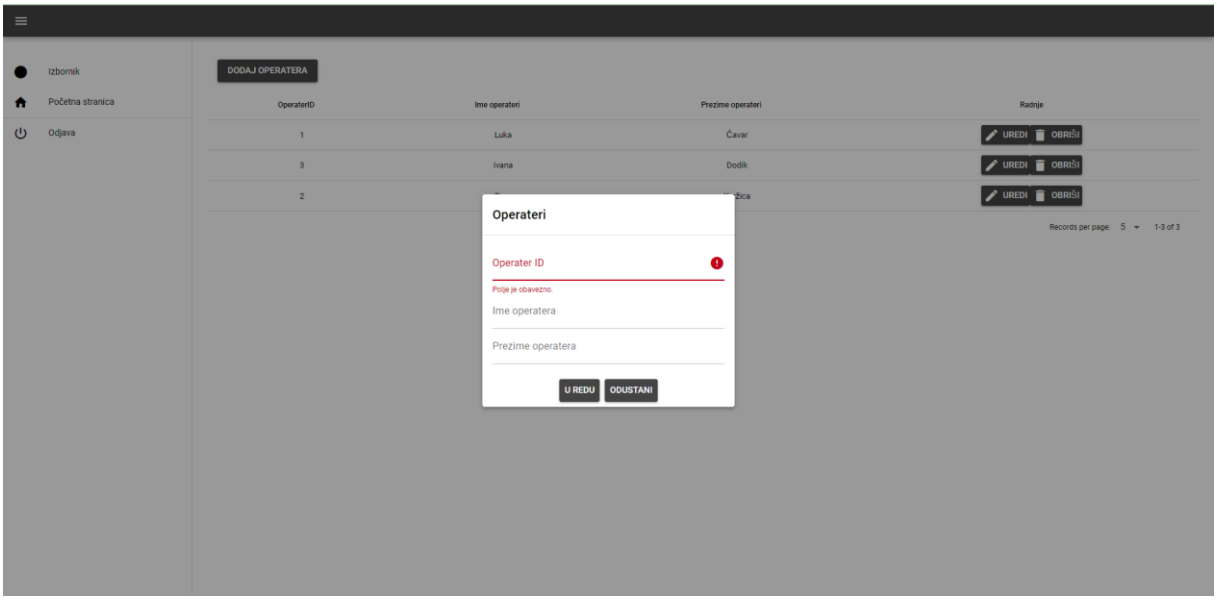
```
            type="text"
            label="Naziv industrije"
            v-model="industrija.Naziv"/>
          <q-input
            ref="Opis"
            type="textarea"
            label="Opis industrije"
            v-model="industrija.Opis"/>
        </q-card-section>
        <q-card-actions align="center">
          <q-btn
            color="#555"
            label="U redu"
            @click="onOKClick"/>
          <q-btn
            color="#555"
            label="Odustani"
            @click="onCancelClick"/>
        </q-card-actions>
      </q-card>
    </q-dialog>
```

U *script* dijelu koda definira se aplikativna logika. Sastoji se od dijelova poput *name*, koji predstavlja jedinstveni naziv komponente, a to je *IndustrijaIndex*.
U *data* dijelu (popis varijabli čija primjena izaziva promjenu na sučelju) nalaze se sljedeće varijable: *openDialog*, *industrija, industrija Model, isMounted, industrije* i *columns.* Varijabla *openDialog* specificirana je za otvaranje i zatvaranje dijaloga za unos i izmjenu podataka. Varijabla *industrija* koriste se za izmjenu podataka u dijalogu. Varijabla u kojoj se nalazi struktura podataka koja se nakon sprema u bazu podataka kao novi dokument naziva se *industrija Model*. Ona se sastoji od sljedećih atributa: IndustrijaID, Naziv industrije i Opis industrije. Varijabla *industrije* označava listu svih podataka iz baze podataka. Ostatak kôda izvodi se na isti način kao i kod prijašnje stranice *RadniNalozi.*

5.4.3 Operateri

Kada korisnik u *Izborniku* pritisne na gumb *Otvori operatere,* preusmjerava se na putanju */Operateri.* Pošto su i *Operateri* dio sučelja *SystemLayot,* stvoreni su kao sadržaj *q-page-containera.* Stranica se sastoji od gumba *Dodaj operatera* i podataka koji su prikazani u organiziranoj tablici. Na kraju tablice, nalaze se dva gumba za izmjenu i brisanje retka tablice – *Uredi* i *Obriši.* Pri pritiskom na gumb *Dodaj operatera* prikazuje se novi prozor(dijalog) za unos podataka koji će nakon unosa biti prikazani u tablici

(Slika 9).



Slika 9: Stranica Operateri

U *template* dijelu datoteke definirano je korisničko sučelje. Ono je definirano s istim varijablama i metodama kao i kod stranica *RadniNalozi* i *Industrija.*
Komponenta *q-input* pomoću koje korisnik unosi podatak. Koriste se više komponenti za unos, a to su: OperaterID, Ime operatera i Prezime operatera. Ove *q-input* komponente imaju postavljenu jedistvenu identifikaciju komponente, zvanu *ref*, pomoću koje se provjerava je li validacija podatka prošla uspješno. Atributom *label* definirane su oznake polja, primjerice *label="OperaterID"*. U prvoj komponenti *q-input, OperaterID,* definirano je pravilo ispravnosti podataka (*:error)*, koje ispisuje poruku „Polje je obavezno." ukoliko korisnik ne upiše nikakvu vrijednosti u to polje. Objekt *operater* koristi se za spremanje unesene komponente; pridodaje mu se atribut, primjerice *OperaterID* na način *operater. OperaterID* gdje se objekt *operater* sastoji od atributa *OperaterID.* Komponenta *q-card-actions* poseban je blok unutar kartice koja sadrži dvije –btn varijable: *U redu* i *Odustani.* Pritiskom na *U redu* gumb poziva se metoda *onOKClick,* dok se pristiskom na *Odustani* gumb poziva *onCancelClick* metoda.

Dio kôda sadržan u datoteci OperaterIndex.vue:

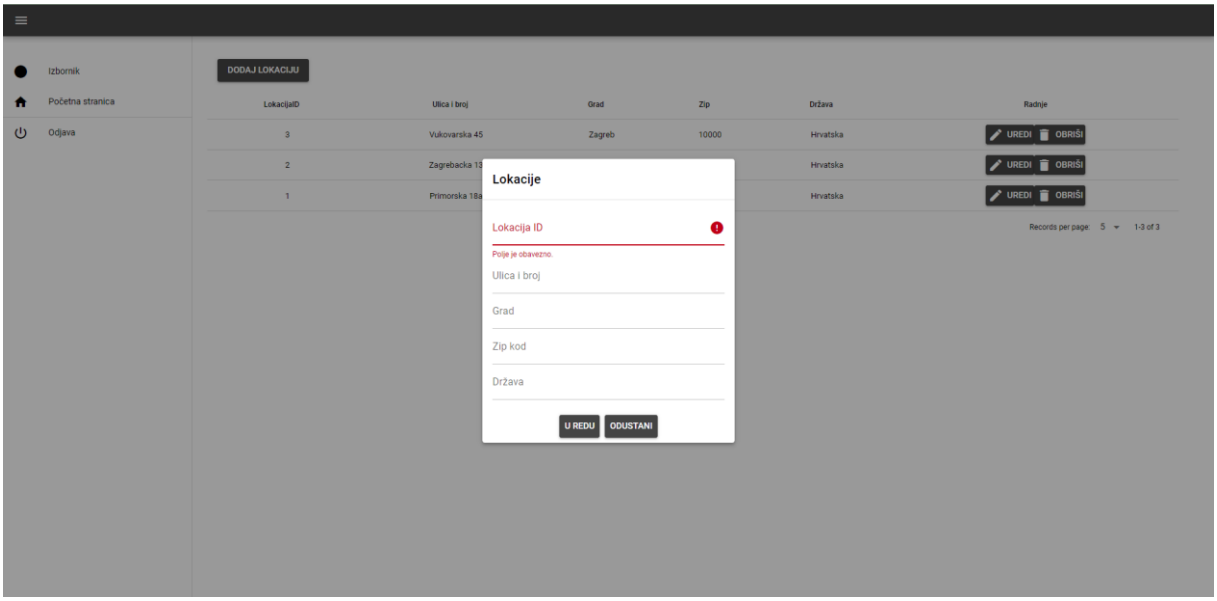
```
<q-dialog
  v-if="openDialog"
  v-model="openDialog">
  <q-card class="q-dialog-plugin">
    <q-card-section>
      <div class="text-h6">Operateri</div>
    </q-card-section>
    <q-separator />
    <q-card-section>
      <q-input
        ref="OperaterID"
        type="number"
        :error="!operater.OperaterID || operater.OperaterID.length === 0"
        error-message="Polje je obavezno."
        label="Operater ID"
        v-model="operater.OperaterID"/>
      <q-input
        ref="Ime"
        type="text"
        label="Ime operatera"
        v-model="operater.Ime"/>
      <q-input
```

```
        ref="Prezime"
        type="text"
        label="Prezime operatera"
        v-model="operater.Prezime"/>
    </q-card-section>
    <q-card-actions align="center">
      <q-btn
        color="#555"
        label="U redu"
        @click="onOKClick"/>
      <q-btn
        color="#555"
        label="Odustani"
        @click="onCancelClick"/>
    </q-card-actions>
  </q-card>
```

U *script* dijelu koda definira se aplikativna logika. Sastoji se od dijelova poput *name*, koji predstavlja jedinstveni naziv komponente, a to je *OperaterIndex*.
U *data* dijelu (popis varijabli čija primjena izaziva promjenu na sučelju) nalaze se sljedeće varijable: *openDialog*, *operater, operaterModel, isMounted, operateri* i *columns.* Varijabla *openDialog* specificirana je za otvaranje i zatvaranje dijaloga za unos i izmjenu podataka. Varijabla *industrija* koriste se za izmjenu podataka u dijalogu. Varijabla u kojoj se nalazi struktura podataka koja se nakon sprema u bazu podataka kao novi dokument naziva se *industrija Model*. Ona se sastoji od sljedećih atributa: OperaterID, Ime operatera i Prezime operatera. Varijabla *operateri* označava listu svih podataka iz baze podataka. Ostatak kôda izvodi se na isti način kao i kod prijašnjih stranica *RadniNalozi* i *Industrija.*

5.4.4 Lokacije

Kada korisnik u *Izborniku* pritisne na gumb *Otvori lokacije,* preusmjerava se na putanju */Lokacije.* Pošto su i *Lokacije* dio sučelja *SystemLayout,* stvoreni su kao sadržaj *q-page-containera.* Stranica se sastoji od gumba *Dodaj lokaciju* i podataka koji su prikazani u organiziranoj tablici. Na kraju tablice, nalaze se dva gumba za izmjenu i brisanje retka tablice – *Uredi* i *Obriši.* Pri pritiskom na gumb *Dodaj lokaciju* prikazuje se novi prozor(dijalog) za unos podataka koji će nakon unosa biti prikazani u tablici (Slika 10).



Slika 10: Stranica Lokacije

U *template* dijelu datoteke definirano je korisničko sučelje. Ono je definirano s istim varijablama i metodama kao i kod stranica RadniNalozi, Industrija i Operateri.
Komponenta *q-input* pomoću koje korisnik unosi podatak. Koriste se više komponenti za unos, a to su: LokacijaID, Ulica, Grad, Zip i Država. Atributom *label* definirane su oznake polja, primjerice *label="LokacijaID"*. U prvoj komponenti *q-input, LokacijaID,* definirano je pravilo ispravnosti podataka (*:error)*, koje ispisuje poruku „Polje je obavezno." ukoliko korisnik ne upiše nikakvu vrijednosti u to polje. Objekt *lokacija* koristi se za spremanje unesene komponente; pridodaje mu se atribut, primjerice *LokacijaID* na način *lokacija. LokacijaID* gdje se objekt *lokacija* sastoji od atributa *LokacijaID.* Komponenta *q-card-actions* poseban je blok unutar kartice koja sadrži dvije –btn varijable: *U redu* i *Odustani.* Ostatak kôda izvodi se na isti način kao i kod prijašnje stranice RadniNalozi.

Dio template kôda sadržan u datoteci LokacijaIndex.vue:

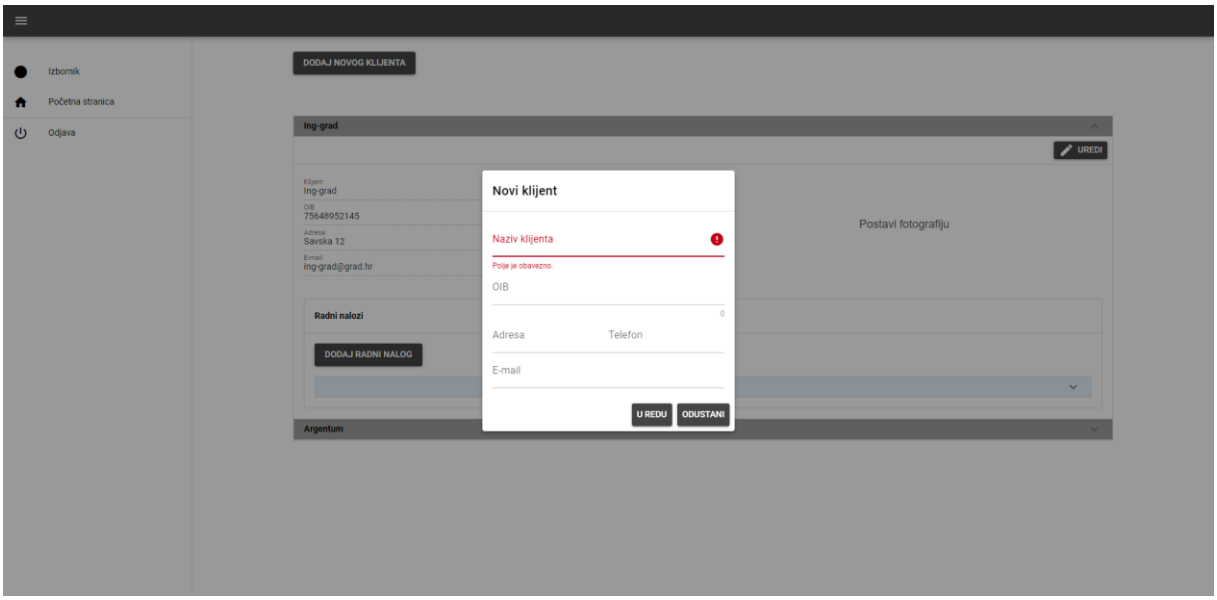
```
<q-card class="q-dialog-plugin">
        <q-card-section>
          <div class="text-h6">Lokacije</div>
        </q-card-section>
        <q-separator />
        <q-card-section>
          <q-input
            ref="LokacijaID"
            type="number"
            :error="!lokacija.LokacijaID || lokacija.LokacijaID.length === 0"
            error-message="Polje je obavezno."
            label="Lokacija ID"
            v-model="lokacija.LokacijaID"/>
          <q-input
            ref="UlicaBroj"
            type="text"
            label="Ulica i broj"
            v-model="lokacija.UlicaBroj"/>
          <q-input
            ref="Grad"
            type="text"
            label="Grad"
            v-model="lokacija.Grad"/>
          <q-input
            ref="Zip"
            type="number"
            label="Zip kod"
            v-model="lokacija.Zip"/>
          <q-input
            ref="Drzava"
            type="text"
            label="Država"
            v-model="lokacija.Drzava"/>
        </q-card-section>
```

U *script* dijelu koda definira se aplikativna logika. Sastoji se od dijelova poput *name*, koji predstavlja jedinstveni naziv komponente, a to je *LokacijaIndex*.
U *data* dijelu (popis varijabli čija primjena izaziva promjenu na sučelju) nalaze se sljedeće varijable: *openDialog*, *lokacija, lokacijaModel, isMounted, lokacije* i *columns.* Varijabla *openDialog* specificirana je za otvaranje i zatvaranje dijaloga za unos i izmjenu podataka. Varijabla *lokacija* koriste se za izmjenu podataka u dijalogu. Varijabla u kojoj se nalazi struktura podataka koja se nakon sprema u bazu podataka kao novi dokument naziva se *lokacijaModel*. Ona se sastoji od sljedećih atributa: LokacijaID, Ulica, Grad, Zip i Država. Varijabla koja označava završno stanje komponente naziva se *isMounted.* Varijabla *lokacije* označava listu svih podataka iz baze podataka. Ostatak kôda izvodi se na isti način

kao i kod prijašnjih stranica RadniNalozi, Industrija i Operateri.

5.4.5 Klijenti

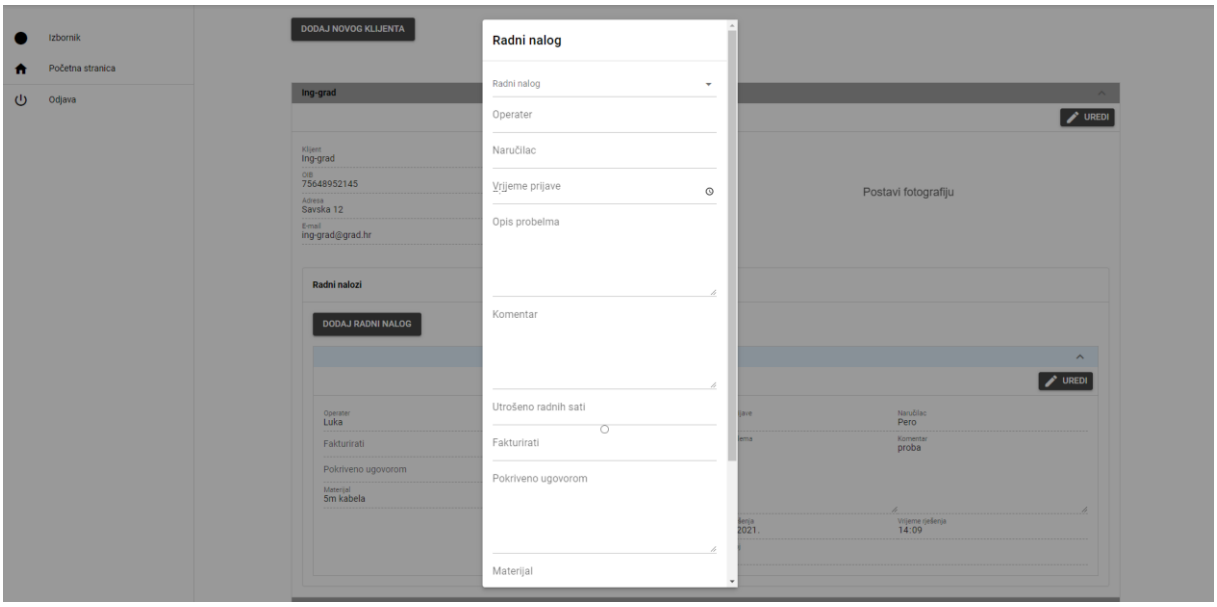
Kada korisnik u *Izborniku* pritisne na gumb *Otvori klijente,* preusmjerava se na putanju */Klijenti.* Stranica se sastoji od gumba *Dodaj klijenta* pri čemu se otvara dijalog za unos podataka o pojedinom klijentu (Slika 11). Također, korisnik ima mogućnost postaviti sliku klijenta.



Slika 11: Stranica Klijenti prilikom klika na gumb Dodaj novog klijenta

Nakon što su uneseni podaci o klijentu, stranica nudi novi gumb *Dodaj radni nalog* te kada ga korisnik pritisne otavara se dijalog za unos podataka o radnom nalogu (Slika 12). U gornjem desnom kutu

kartice u kojoj se nalaze podaci o klijentu, nalazi se gumb za izmjenu podataka – *Uredi.* Mogućnost izmjene podataka pomoću gumba *Uredi* također je dostupan i u kartici s podacima o radnim nalozima.



Slika 12: Stranica Klijenti prilikom klika na gumb Dodaj radni nalog

U *template* dijelu datoteke definirano je korisničko sučelje. Kod će se objašnjavati redoslijedom kako se definiraju blokovi i komponente. U *div* bloku definirani su elementi sučelja, gdje sadržaj sadrži marginu *q-pa-lg* sa svih strana, prikazuje se u obliku retka *row* te je horizontalno poravnat *justify-center.* U drugom *div* bloku definiran je sadržaj koji se prikazuje kao stupac *column,* čija je maksimalna širina 1300 piksela. Komponenta *q-btn* definira gumb *Dodaj klijenta*, čijim se pritiskom na gumb poziva metoda *onNewKlijent.*
U *div* bloku kôda, koji je prikazana ispod teksta, definiran je atribut *v-for.* Blok se ponavlja onoliko puta koliko ima elemenata u u listi *klijenti.* Elementi su dostupni preko varijable *Klijent,* dok je indeks elemenata dostupan preko varijable *KlijentIndex.* Svaki element ima jednistvenu vrijednost atributa *key,* koja je postavljena pomoću *UIDKlijent.*

```
<div
  v-for="(Klijent, KlijentIndex) in klijenti"
  :key="Klijent.UIDKlijent">
```

U komponenti *q-expansion-item* biti će sadržaj prijašnjeg bloka *div.* Komponenta ima funkciju skrivanja svojeg sadržaja prilikom klika na nju. Sadržaj se definira unutar komponente *q-card,* te se više ovih komponenti može grupirati pomoću atributa *group* koji ima istu vrijednost za sve komponente, a ona je u ovom slučaju *klijenti.* Prva *q-expansion-item* je postavljena da joj *defaultno* stanje bude „0“ , tj. uvjet *KlijentIndex == 0,* kako bi se otvarala čim se stranica otvori.
Komponenta *q-card-actions* poseban je blok unutar *q-card* komponente u kojem se nalazi *q-btn* gumb *Uredi.* Pritiskom na njega poziva se metoda *onUpdateKlijent*.
Komponenta *q-card-section* poseban je blok unutar *q-card* u kojoj se nalaze *q-input* komponente za unos podataka. One su pomoću *v-model* atributa povezane s odgovarajućim atributom unutar varijable *Klijent.* Kako bi se *q-input* komponente prikazale pomoću redaka i stupaca, kreirana su dva *div* bloka; jedan za prikaz o retku, a drugi koji sadržaj iz jednog retka prikazuje unutar kolone.
Komponenta koja se korirsti za postavljanje slike zove se *croppa.* Pomoću definiranog *v-modela* za sliku *Klijent.photo* upisivat će se osnovni podaci o izabranoj slici. Postavljena je url putanja slike koja se čuva

unutar atributa *urlimage* varijable *Klijent.* Metoda *removeImage* poziva se preko događaja *@image-remove* te se pomoću nje briše slika ako je potrebno. Također su postavljeni događaji *@file-type-mismatch* i *@file-size-exceed* koje pozivaju metodu *onError* pomoću koje se ispisuje poruke o greški prilikom postavljanja slike. Varijabla *uploadButton* prikazuje je li gumb za postavljanje vidljiv. Postavljena je na *true* na događaju @file-choose. Klikom na gumb *Postavi fotografiju*, čija vidljivost ovisi o varijabli *uploadButton,* poziva se metoda *uploadCroppedImage.* Ukoliko je korisnik na mobitelu i želi preko njega postaviti fotografiju, klikom se poziva metoda *openCamera* kako bi korisnik mogao odabrati željenu fotografiju.
Komponenta *q-dialog* postavljen je na isti način kako su postavljene i *q-dialog* komponente na prijašnjim stranicama, no u ovom slučaju jedan je novi dodatak, a to je *q-select* komponenta. Pomoću nje se postavljaju padajuće liste iz kojih korisnik može odabrati podatak. Koristimo je kako bi izabrali radni nalog iz liste radnih naloga. Izbor vrijednosti koji će se prikazivati u listi izbora definirano je preko atributa *options,* a postavljeno je da će se prikazivati izbori iz liste *nalozi.* Dodijeljen joj je *v-model Klijentradninalog.radninalog.* Također, postavljeno je da se prikazuje vrijednost *option-label* atributa *type*, a vrijednost *option-value* je *UIDradninalog.*

U *script* dijelu datoteke definirana je aplikativna logika. Na početku se uvozi (*import)* objekt *component* koji se zove *croppa*. On se uvozi iz datoteke modula *vue-croppa* kojeg je potrebno prije pomoću naredbe instalirati u projekt te se koristi kao komponenta za postavljanje slike u *template* dijelu datoteke. Jedinstveni naziv komponente definira se pomoću *name,* a u ovoj datoteci to je *KlijentiIndex.*
Dio kôda sadržan u datoteci KlijentIndex.vue:

```
<q-card class="q-dialog-plugin">

        <q-card-section>
          <div class="text-h6">
            Radni nalog
          </div>
        </q-card-section>
        <q-separator />
        <q-card-section>
          <q-select
            label="Radni nalog"
            dense
            v-model="Klijentradninalog.radninalog"
            :options="nalozi"
            option-value="RadniNalogID"
            option-label="type"/>
```

U *data* dijelu *script* kôda nalaze se varijable čija promjena izaziva promjenu na korisničkom sučelju. Koristile su se varijable *openKlijentDialog* i *openKlijentradninalogDialog* koje definiraju otvaranje i zatvaranje dijaloga za unos ili izmjenu podataka o klijentima i njihovim radnim nalozima. Varijabla koja predstavlja listu klijenata u bazi podataka zove se *klijenti,* dok je *Klijent* varijabla u kojoj se nalazi klijent koji je spreman za unos i izmjenu podataka pomoću dijaloga. Struktura koja će se spremiti u kolekciju baze podataka i koja se koristi pri izradi novog klijenta zove se *KlijentModel.* U varijabli *currentKlijent* nalazi se trenutno izabrani klijent, dok se u varijabli *Klijentradninalog* nalazi izabran radni nalog u klijentima te se koristi pri unosu i izmjeni podataka novog radnog naloga preko dijaloga. Struktura koja će se spremiti u kolekciju baze podataka i koja se koristi pri izradi novog radnog naloga u klijentima. Varijabla *uploadButton* koristi se za pokazivanje i skrivanje gumba za postavljanje slike na server aplikacije. Varijabla *initalImage* koriste se u slučaju da korisnik želi postaviti sliku pomoću kamere mobilnog uređaja. Varijabla *nalozi* predstavlja listu radnih naloga te se koristi unutar *q-select*

komponente kako bi dohvatila ponuđene izbore radnih naloga iz liste.
Kada se pozove varijabla *mounted***,** stvara se referenca na kolekciju *Klijent* u firestore bazi podataka preko koje se dolazi pomoću javnog dostupnog atributa *this.$db* definiranog u datoteci *RadniNalogSetup.vue.* Pomoću firestore metode *get*, koja nam vraća sve retke, tj. dokumente iz kolekcije *lokacija,* referenca se sprema u varijablu *colectionRef*. Svakom vraćenom retku dodaju se dva atributa, a to su *Klijentradninalog* i *photo.* Nakon toga, svaki redak stvara referencu na njegovu podkolekciju preko koje se učitavaju dokumenti podkolekcije. U varijablu *KlijentradninalogFromDB* ubacuje se svaki redak iz podkolekcije. Ona se kasnije stavlja u atribut *Klijentnalozi* varijable *KlijentFromDB*. Zatim, koristeći firestore metodu *push*, varijabla *KlientFromDB* ubacuje se u varijablu *klijenti* te se ona koristi u template dijelu datoteke kako bi omogućila prikaz podataka o radnim nalozima. Također, učitavaju se i podaci o radnim nalozima iz kolekcije *nalozi.*
Pritiskom na gumb *Dodaj novog klijenta* poziva se metoda *onNewKlijentRow.* Ona u varijablu *Klijent* stvara novi objekt koji će poprimiti strukturu *KlijentModel.* Varijabla *openKlijentDialog* se postavlja na vrijednost *true* čime se otvara dijalog za unos podataka o klijentu.

Pritiskom na gumb *Dodaj radni nalog* poziva se metoda *onNewKlijentradninalogRow.* Ona u varijablu *Klijentradninalog* stvara novi objekt koji će poprimiti strukturu *KlijentradninalogModel.* Varijabla *opneKlijentDialog* postavlja se na *true* čime se otvara dijalog za unos radni naloga unutar klijenta. Pritiskom na gumb *U redu* u dijalogu, poziva se metoda *onOKKlijentDialogClick*. Metoda provjera jesu li podaci uspješno prošli validaciju specifirana na *q-input* komponenti. Ako nije došlo do pogreške u komponentama, referenca se sprema u varijablu *collectionRef* prema kolekciji *klijent.* Nakon toga, radi se provjera je li došlo do novog retka. Ukoliko je, tada se pomoću firestore metode za dodavanje novih dokumenata dodaje novi dokument u kolekciju *klijent.* Prilikom uspješnog dodavanja, u atribut *lokacija.UIDKlijent* se sprema identifikacija dokumenta koju firestore automatski postavlja. Referenca se dodaje prema novom spremljenom dokumentu kako bi se u njemu ažurirao atribut *UIDKLijent* pomoću metode *update.* Ukoliko je ažuriranje uspješno obavljeno, dodaje se novi atribut *Klijentnalozi* u listu *klijent* čime se ažurira i sama tablica. Varijabla *openDialog* postavlja se na *false* kako bi se zatvorio dijalog prilikom uspješnog izvršenja svih naredbi. Ako je došlo do neke greške prilikom ažuriranja ili dodavanja dokumenta, ispisuje se poruka „Pogreška pri dodavanju dokumenta“. Ukoliko nije došlo do novog retka/unosa, referenca se dohvaća prema dokumentu. Pomoću firestore metode dokumenta *set,* stari dokument zamjenjuje se novim. Prilikom uspješnog postavljanja, lokacija je poslana u bazu podataka pronalazi se u listi lokacija te se ažurira. Varijabla *openDialog* postavlja se na *false* kako bi se zatvorio dijalog, a ukoliko je došlo do pogreške u izvođenju koda ispisuje se poruka pogreške.
Pritiskom na gumb *U redu* unutar dijaloga za unos podataka o radnom nalogu, poziva se metoda *onOKKlijentradninalogDialogClick.* U njoj se stvara referenca na podkolekciju *Klijentradninalog* iz kolekcije *Klijenti*. Ostale funkcije i varijable su iste kao u metodi *onOKKlijentDialogClick.*
Pritiskom na gumb *Odustani* unutar bloka za unos podataka o novom klijentu, poziva se metoda *onCancelKlijentDialogClick.* Ona postavlja varijablu *openDialog* na *false* te se time dijalog zatvara. Sukladno s time, pritiskom na gumb *Odustani* unutar dijaloga za unos radnog naloga klijenta, poziva se metoda *onCancelKlijentradninalogDialogClick* koja postavlja varijablu *openDialog* na *false* te se time dijalog zatvara.
Pritiskom na gumb za izmjenu retka *Uredi*, poziva se metoda *onUpdateKlijent*. On funkcionira na način da se u varijablu *Klijent* dodaje novi objekt strukture *KijentModel* te u njega prepisuje podatke retka dobiveni kroz argument metode. Sukladno time, na isti način funkcionira i drugi gumb *Uredi* koji poziva

metodu *onUpdateKlijentradninalog* za uređivanje podataka o radnimm nalozima klijenta. Nakon dodavanja novog objekta, varijabla *openDialog* postavlja se na *true* čime se otvara dijalog. Pritiskom na gumb *Postavi fotografiju* poziva se metoda *uploadCroppedImage.* Ona provjerava postoji li slika u atributu *photo* unutar kolekcije *Klijent*. Ukoliko postoji, poziva se metoda *generateBlob* koja generira sliku i dostavlja ju na server. Pomoću firebase *storage* metode *ref()* stvara se referenca prema serveru do koje se dolazzi preko varijable *$storage* koja je definirana u datoteci *RadniNalogSetup.vue*. Pomoću firebase metode *child* dostavlja se slika na server. Ukoliko je postavljanje slike bilo uspješno, tada se pomoću firebase metode *getDownloadURL* dolazi do njezine URL adrese. Ako je dohvaćanje URL adrese prošlo uspješno, tada se ona sprema u varijablu *Klijent.urlImage* koja se koristi za prikaz slike pomoću komponente *croppa.*
Metoda koja se poziva za brisanje slike naziva se *removeImage.* Ukoliko je URL adresa slike postavljena u objekt *Klijent,* postavlja se referenca serveru i prema slici na tom serveru. Pomoću firebase metode *delete* briše se datoteka, te ukoliko je brisanje prošlo uspješno, atribut *urlImage* od objekta *Klijent* postavlja se na *null* i takav se sprema u bazu podataka.

5. Zaključak

Web aplikacija za stvaranje i upravljanje radnim nalozima je vrlo korisna aplikacija u poslovnom smislu. Tvrtke, poduzeća ili organizacije koje žele uštedjeti vrijeme i novac na izradu papirologije i efikasno upravljati svojim radnim nalozima, koristit će aplikacije poput ove koja je predstavljena i objašnjena u radu. Jednostavna je za korištenje te sadrži elemente i dizajn koji jasno vode korisnika kroz aplikaciju. S druge strane, održavanje ove aplikacije nije zahtjevno. Podaci s aplikacije se spremaju u organiziranu bazu podataka koja ih prikazuje pregledno i detaljno, kako bi korisnik imao uvid u svoje poslovanje. Također, sigurnost je jako bitna u poslovanju tvrtki. Ne treba brinuti o tome jer su podaci pohranjeni na sigurnoj bazi podataka, a pristupu sustavu za upravljanje imaju samo određeni korisnici i administratori. Aplikacije poput ove, s obzirom na povećanje korištenja web i mobilnih aplikacija, su nužne u današnje doba. U slučaju da korisniku nešto iskrsne na poslu ili da nije pokraj računala, aplikacija će mu pomoći jer joj se može pristupiti s bilo kojeg uređaja, na bilo kojem mjestu i u bilo koje vrijeme. Pogotovo je korisna u vrijeme globane pandemije, gdje ljudi rade od kuće preko računala i nisu u kontaktu s vanjskim svijetom - potrebna im je aplikacija pomoću koje će voditi evidenciju o poslovanju tvrtke te im je dostupna i „kod kuće“.

6. Popis literature

[1] https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app zadnji pristup: 15.08.2021
[2] https://blog.stackpath.com/web-application/ zadnji pristup: 16.08.2021
[3] https://www.indeed.com/career-advice/career-development/what-is-web-application zadnji pristup: 17.08.2021
[4] Wakil, K., & Jawawi, D. N. (2019). Intelligent Web Applications as Future Generation of Web Applications. *Scientific Journal of Informatics*, *6*(2), 214.
[5] https://www.onupkeep.com/ zadnji pristup: 18.08.2021
[6] https://www.hashmicro.com/blog/the-benefits-of-web-based-apps/ zadnji pristup: 18.08.2021

[7] https://www.onupkeep.com/whcapterraat-is-a-work-order zadnji pristup: 19.08.2021
[8] https://www.landport.net/top-5-benefits-of-online-work-order-management/ zadnji pristup: 19.08.2021
[9] https://mydatascope.com/blog/en/work-orders-everything-you-need-to-know-about-them/
[10] https://www.capterra.com/work-order-software/ zadnji pristup: 19.08.2021
[11] https://try.wrike.com/work-order-capterra/?utm_medium=cpc&utm_campaign=work_order&utm_content=listing&utm_source=capterra&dclid=CKXFgtX5y_ICFW-C_QcdVzMDMg zadnji pristup: 20.08.2021
[12] https://web.dev/what-are-pwas/ zadnji pristup: 20.08.2021
[13] https://www.ezofficeinventory.com/solutions/work-order-software?r=capterra_work_order&utm_source=capterra zadnji pristup: 21.08.2021
[14] https://quasar.dev/introduction-to-quasar zadnji pristup: 21.08.2021
[15] https://javascript.info/intro zadnji pristup: 22.08.2021
[16] https://www.javatpoint.com/what-is-html zadnji pristup: 22.08.2021
[17] https://www.javatpoint.com/what-is-css zadnji pristup: 23.08.2021
[18] https://www.guru99.com/introduction-to-database-sql.html zadnji pristup: 23.08.2021
[19] https://www.geeksforgeeks.org/distributed-database-system/ zadnji pristup: 24.08.2021
[20] https://www.ibm.com/cloud/learn/what-is-cloud-database zadnji pristup: 24.08.2021
[21] https://www.nginx.com/resources/glossary/web-server/ zadnji pristup: 25.08.2021
[22] https://firebase.google.com/docs/hosting zadnji pristup: 25.08.2021
[23] https://mydatascope.com/blog/en/work-orders-everything-you-need-to-know-about-them/ zadnji pristup: 26.08.2021
[24] https://www.onupkeep.com/what-is-a-work-order zadni pristup: 25.08.2021
[25] https://www.codecademy.com/articles/what-is-an-ide zadnji pristup: 22.08.2021
[26] https://www.techopedia.com/definition/14384/software-framework zadnji pristup: 29.08.2021
[27] https://www.ibm.com/cloud/learn/nosql-databases zadnji pristup: 27.08.2021

7. Popis slika