Razvoj web aplikacije za kozmetičke salone

Radović, Laura

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Laura Radović

Razvoj web aplikacije za kozmetičke salone

Završni rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, rujan 2024.

Zadatak

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija
uniri

Rijeka, 30.4.2024.

Zadatak za završni rad

Pristupnik/ica: Laura Radović

Naziv završnog rada: Razvoj web aplikacije za kozmetičke salone

Naziv završnog rada na engleskom jeziku: Development of a web application for beauty salons

Sadržaj zadatka: Zadatak završnog rada je dizajnirati i izraditi web aplikaciju za kozmetičke salone. U radu će se detaljno opisati svi postupci korišteni prilikom dizajniranja i izrade web aplikacije te svi alati koji su se pritom koristili.

Mentor/ica
Titula i ime mentora/ice

Nadimanć Nikola

_____

Voditelj za završne radove
Izv. prof. dr. sc. Miran Pobar

_____

Zadatak preuzet: 30.4.2024.

Laura Radović
(potpis pristupnika/ice)

Radmile Matejčić 2,
51000 Rijeka, Hrvatska

T: +385 (0)51/584-700
E: ured@inf.uniri.hr
www.inf.uniri.hr

OIB: 64218323816
IBAN: HR1524020061400006966

Sažetak

Ovaj završni rad prikazuje postupak izrade web aplikacije za kozmetički salon "Glamour" koja je dizajnirana za prikaz i upravljanje sadržajem kozmetičkog salona. Korištenjem tehnologija kao što su HTML, CSS, JavaScript te Vue.js, aplikacija pruža intuitivno korisničko sučelje koje omogućuje administratorima jednostavno upravljanje fotografijama i uslugama putem Google Firebase platforme. Proces razvoja uključuje planiranje, instalaciju potrebnih alata, te implementaciju ključnih funkcionalnosti, uključujući prijavu korisnika i administrativni dio stranice.

Ključne riječi: web aplikacija; Vue.js; JavaScript; HTML; CSS; Google Firebase; Figma; kozmetički salon

Sadržaj

1. Uvod

U ovom završnom radu prikazan je postupak izrade web aplikacije za prezentaciju salona *Glamour*. Aplikacija je izrađena korištenjem HTML-a, CSS-a, JavaScript-a te Vue.js-a, modernog JavaScript okvira (engl. *framework*) za izgradnju korisničkih sučelja. Za pohranu i upravljanje slikama koristi se Google Firebase.

Rezultat ovog diplomskog rada je "Glamour" web aplikacija koja administratoru omogućuje dodavanje, pregled i upravljanje fotografijama, prikazan je popis usluga koje su u ponudi korisniku, također je dostupna galerija gdje korisnici mogu pregledati slike usluga. Ideja za izradu ove aplikacije proizašla je iz potrebe za web aplikacijom salona za uljepšavanje. Aplikacija olakšava tvrtkama u industriji ljepote da prezentiraju svoje proizvode na atraktivan i korisnicima pristupačan način, što može značajno doprinijeti povećanju klijentele i poboljšanju korisničkog iskustva.

Aplikacija nudi mogućnost prijave administratora, omogućujući im da dodaju i uklanjaju fotografije proizvoda koje žele prikazati. Administrator također može unositi podatke o uslugama, kao što su naziv usluge i cijena. Osim unosa podataka, aplikacija omogućuje uređivanje i brisanje postojećih podataka o proizvodima. Na taj način, tvrtke mogu jednostavno ažurirati informacije i osigurati da su podaci uvijek točni i relevantni.

Osim osnovnih funkcionalnosti, aplikacija također omogućuje pohranu podataka na Firebase, što osigurava sigurnost i pouzdanost podataka, te omogućuje pristup podacima s bilo kojeg uređaja u bilo kojem trenutku. Ovo rješenje nudi jednostavnost korištenja i efikasnost u upravljanju digitalnim sadržajem, što je ključno za suvremene poslovne potrebe.

Izrada ove aplikacije omogućuje stjecanje praktičnog iskustva u korištenju najnovijih web tehnologija te pruža vrijednu platformu za unapređenje poslovnih procesa u industriji ljepote.

1. Opis tehnologija

Ovo poglavlje sadrži opis alata, programskih jezika i baze podataka korištenih u izradi web aplikacije za kozmetički salon "Glamour".

1.1 Figma

Figma je web aplikacija namijenjena za dizajniranje korisničkih sučelja, s dodatnim mogućnostima rada izvan mreže putem desktop aplikacija dostupnih za macOS i Windows (Figma, 2022). Također, Figma ima mobilnu aplikaciju za Android i iOS koja omogućuje pregledavanje i interakciju s prototipovima u stvarnom vremenu na mobilnim uređajima. Njene funkcionalnosti su fokusirane na dizajn korisničkog sučelja (UI) i korisničkog iskustva (UX), s naglaskom na timsku suradnju u stvarnom vremenu kroz alate za vektorsku grafiku i izradu prototipova.

Jedna od najvažnijih značajki Figme je mogućnost istovremenog dijeljenja datoteka među suradnicima. Za razliku od tradicionalnih aplikacija kao što su *Sketch* i *Photoshop,* gdje dizajneri moraju ručno dijeliti statične slike, Figma omogućuje korisnicima dijeljenje linkova na datoteke, što smanjuje vrijeme potrebno za komunikaciju i omogućuje pristup najnovijim verzijama datoteka bez potrebe za stalnim ažuriranjem.

1.2. HTML (HyperText Markup Language)

HTML, skraćenica za HyperText Markup Language, koristi se za izradu internetskih stranica koristeći jezik za označavanje. Kombinacija hiperteksta i jezika za označavanje omogućava definiranje veza između stranica, dok jezik za označavanje određuje strukturu tekstualnog dokumenta. Većina jezika za označavanje, uključujući HTML, čitljiva je ljudima i omogućava stroju manipulaciju tekstom prema specificiranim oznakama.

HTML je osnovni jezik za označavanje koji preglednici koriste za rad s tekstom, slikama i drugim sadržajem kako bi ih prikazali u traženom formatu. Razvio ga je Tim Berners-Lee 1991. godine, a prva standardna verzija, HTML 2.0, objavljena je 1995. godine (W3C, 2024).

HTML koristi unaprijed definirane oznake i elemente koji govore pregledniku kako pravilno prikazati sadržaj. Ključni elementi HTML strukture uključuju *doctype* deklaraciju, <*html*>, <*head*> i <*body*> oznake.

Doctype deklaracija <*!DOCTYPE html*> identificira dokument kao HTML dokument, dok oznaka <html> označava korijenski element HTML-a, unutar kojeg se nalaze svi ostali elementi. Unutar oznake <*head*> nalaze se pozadinski elementi kao što su <*style*>, <*title*>, <*base*>, <*noscript*>, <*script*>, <*meta*> i <*link*>, koji nisu vidljivi na prednjem dijelu stranice. Oznaka <*body*> sadrži sav vidljivi sadržaj stranice koji se prikazuje u pregledniku.

HTML dokument se može izraditi u bilo kojem uređivaču teksta, a potrebno je spremiti datoteku s ekstenzijom .html ili .htm kako bi se mogla otvoriti kao internetska stranica u pregledniku.

Značajke HTML-a uključuju jednostavnost učenja i korištenja, neovisnost o platformi, mogućnost dodavanja slika, videozapisa i audio zapisa na stranicu, te podršku za hipertekst.

HTML omogućava izradu osnovnih internetskih stranica i često se koristi u kombinaciji s CSS-om i JavaScriptom za stvaranje dinamičkih i vizualno privlačnih stranica.

Međutim, HTML ima i svoje nedostatke: koristi se samo za statične stranice, za dinamičke stranice potrebni su drugi jezici, pisanje velikih količina koda za jednostavne stranice može biti naporno, a sigurnosne značajke nisu na visokoj razini. Unatoč tome, HTML je temeljni jezik za razvoj internetskih stranica i neophodan je za svakog web programera.

1.3. CSS (Cascading Style Sheets)

CSS, kratica za Cascading Style Sheets, predstavlja jednostavan jezik dizajniranja koji značajno olakšava proces izrade internetskih stranica. Prema tome, CSS omogućuje oblikovanje stranica neovisno o HTML-u, što uključuje definiranje boja, fontova, razmaka i drugih vizualnih elemenata. Na taj način, dizajneri imaju mogućnost prilagoditi izgled stranice prema svojim željama, dok programeri mogu detaljno odrediti ponašanje elemenata, uključujući njihov raspored unutar preglednika (Smith, 2024).

CSS koristi skup pravila, za razliku od HTML-a koji koristi oznake. Iako je jednostavan za učenje i razumijevanje, CSS pruža moćnu kontrolu nad prezentacijom HTML dokumenata. Jedna od glavnih prednosti CSS-a je ušteda vremena jer se može napisati jedan stil i koristiti ga na više HTML stranica. Održavanje je također jednostavnije, jer promjena stila na jednom mjestu automatski ažurira sve povezane elemente na svim stranicama.

CSS se smatra čistom tehnikom kodiranja, što znači da preglednici lako čitaju njegov sadržaj. U usporedbi s HTML-om, CSS nudi naprednije stilove i širi raspon atributa, čime se postiže bolji vizualni izgled stranice. Osim toga, CSS omogućuje izvanmrežno pregledavanje internetskih aplikacija pomoću lokalne predmemorije, što omogućava korisnicima pregled stranica čak i bez internetske veze.

CSS se sastoji od stilskih pravila koja preglednik tumači i primjenjuje na odgovarajuće elemente u dokumentu. Svako stilsko pravilo uključuje selektor i blok deklaracije. Selektor identificira HTML element koji će se stilizirati, dok deklaracijski blok sadrži jednu ili više deklaracija odvojenih točkom i zarezom. Svaka deklaracija sastoji se od CSS svojstva i njegove vrijednosti, odvojenih dvotočkom. Deklaracije završavaju točkom i zarezom, a blokovi deklaracije su okruženi vitičastim zagradama.

CSS selektori koriste se za odabir HTML elemenata prema njihovom nazivu, ID-u, klasi, atributima i drugim kriterijima. Postoji nekoliko vrsta selektora: univerzalni, elementni, potomak, ID, klasni i grupni selektori. Univerzalni selektor odabire bilo koji element, elementni selektor odabire elemente po nazivu, a potomak selektor primjenjuje stil samo na elemente unutar određenog elementa. ID selektor koristi jedinstveni ID atribut HTML elementa, dok klasni selektor odabire elemente prema klasi. Grupni selektori omogućuju grupiranje elemenata s istim stilovima kako bi se kod optimizirao i minimizirao.

Korištenjem CSS-a, internetske stranice postaju estetski ugodnije, preglednije i lakše za održavanje. Također omogućuje bržu i jednostavniju prilagodbu izgleda stranica, čime se poboljšava korisničko iskustvo i olakšava rad dizajnera i programera.

1.4. JavaScript

JavaScript je jednostavan, višeplatformski i interpretirani programski jezik, često korišten za razvoj dinamičkih internetskih stranica (W3Schools, 2023). Omogućuje kreiranje interaktivnih elemenata i omogućuje programiranje kako na strani klijenta, tako i na strani poslužitelja. Ovaj skriptni jezik poznat je po svojoj primjeni u razvoju web aplikacija, ali se koristi i u mnogim drugim okruženjima izvan preglednika.

JavaScript podržava imperativno i deklarativno programiranje te dolazi sa standardnom knjižnicom objekata kao što su Array, Date i Math, kao i osnovnim skupom jezičnih elemenata poput operatora, kontrolnih struktura i izjava. Kada se koristi za razvoj na strani klijenta, JavaScript omogućuje manipulaciju preglednikom i Document Object Modelom (DOM). Ovo omogućuje aplikacijama interakciju s korisnicima putem HTML obrazaca, reagiranje na događaje poput klikova mišem ili unosa podataka. Popularne klijentske knjižnice uključuju AngularJS, ReactJS i VueJS.

Na strani poslužitelja, JavaScript omogućuje komunikaciju s bazama podataka i manipulaciju datotekama, što je posebno korisno u aplikacijama koje zahtijevaju kontinuitet informacija. Jedna od najpopularnijih platformi za serverski JavaScript je Node.js, koja omogućuje izradu skalabilnih mrežnih aplikacija.

JavaScript može biti uključen u HTML dokumente kao interni ili vanjski skript. Interni JavaScript se piše unutar <script> oznaka direktno u HTML datoteci, dok se vanjski JavaScript nalazi u zasebnim .js datotekama koje se povezuju unutar <head> oznake HTML-a.

JavaScript je razvijen 1995. godine od strane Brendana Eicha iz Netscapea (W3Schools, 2024) Dizajniran je da radi kao skriptni jezik unutar preglednika, omogućujući dinamične interakcije koje su prije bile nemoguće sa statičnim HTML stranicama. Jedna od ključnih prednosti JavaScripta je njegova sposobnost manipulacije DOM-om, čime se omogućuje dinamično mijenjanje sadržaja stranice bez ponovnog učitavanja.

Funkcije u JavaScriptu su objekti, što znači da mogu imati svojstva i metode te se mogu prosljeđivati kao argumenti drugim funkcijama. JavaScript također omogućuje rad s datumima i vremenom, te provjeru valjanosti obrazaca kreiranih pomoću HTML-a. Za razliku od mnogih drugih jezika, JavaScript ne zahtijeva kompajler, što olakšava razvoj i testiranje koda.

Primjene JavaScripta su široke, uključujući razvoj web stranica, web aplikacija, serverskih aplikacija, igara, aplikacija za pametne satove, umjetničkih instalacija, strojnog učenja i mobilnih aplikacija. Korištenjem kombinacija JavaScripta s HTML5, moguće je kreirati složene i interaktivne igre, dok se pomoću Node.js-a može ostvariti serverski razvoj.

Ograničenja JavaScripta uključuju sigurnosne rizike, performanse, složenost koda i slabo rukovanje pogreškama. Cross-site scripting napadi, gdje zlonamjerni JavaScript kod dohvaća podatke iz preglednika, česti su sigurnosni problemi. Performanse JavaScripta su također ograničene u usporedbi s tradicionalnim programskim jezicima, jer je relativno spor za složene operacije.

Iako je JavaScript lagan za implementaciju s minimalističkom sintaksom, potrebno je temeljito poznavanje programskih koncepata za pisanje naprednih skripti. Nedostatak statičkog tipiziranja može uzrokovati pogreške koje nije moguće otkriti prije izvođenja koda. Unatoč tim

ograničenjima, JavaScript ostaje jedan od najpopularnijih i najmoćnijih jezika za razvoj dinamičkih web stranica i aplikacija.

1.5. Vue.js

Vue.js je JavaScript okvir koji je posebno pogodan za razvoj jednostraničnih aplikacija (SPA). Korištenjem Vue.js-a, aplikacije se mogu učitati samo jednom, a sadržaj se dinamički ažurira bez potrebe za ponovnim učitavanjem cijele stranice (ADCI Solutions, 2023). Ovo ga čini idealnim za moderne web aplikacije koje zahtijevaju brzo i učinkovito osvježavanje sadržaja.

Interoperabilnost Vue.js-a omogućuje njegovu upotrebu u kombinaciji s raznim drugim knjižnicama i alatima, čime se može lako integrirati u postojeće projekte. Također, podržan je u svim glavnim preglednicima, uključujući Chrome, Firefox, Internet Explorer i Safari, što olakšava njegovu primjenu u širokom rasponu okruženja.

Jedna od glavnih prednosti Vue.js-a je njegova prilagodljivost. Programeri ga mogu koristiti u onolikoj mjeri koliko je potrebno za njihov projekt, bilo da se radi o malim dijelovima web stranice ili cijeloj aplikaciji. Uz to, Vue.js koristi komponente, koje omogućuju stvaranje prilagođenih HTML elemenata koji se mogu ponovno upotrijebiti u različitim dijelovima aplikacije, čime se povećava modularnost i olakšava održavanje koda.

Još jedna značajna karakteristika Vue.js-a su prijelazi, koji omogućuju glatke animacije prilikom dodavanja ili uklanjanja komponenti iz Document Object Modela (DOM). To pruža bolje korisničko iskustvo kroz vizualne efekte i prijelaze između različitih stanja aplikacije.

Vue.js također dolazi s vrlo detaljnom dokumentacijom, koja pokriva sve aspekte okvira, omogućujući programerima da brzo nauče i počnu koristiti Vue.js, bez obzira na njihovu razinu iskustva. Što se tiče instalacije, Vue.js se može integrirati u projekt na nekoliko načina: putem CDN-a jednostavnim dodavanjem <script> oznake u HTML datoteku, putem npm-a (Node Package Manager), što je najčešći način za moderne JavaScript aplikacije, ili putem Vue CLI (Command Line Interface), koji pruža napredne opcije za postavljanje Vue.js projekata, uključujući predloške i integraciju s alatima poput Webpacka (Vue.js, 2023)

Sve ove značajke čine Vue.js moćnim alatom za razvoj modernih web aplikacija, omogućujući fleksibilnost i lakoću upotrebe za programere.

1.6. Node.js

Node.js je okruženje otvorenog koda koje omogućava izvršavanje JavaScript koda izvan preglednika. Iako nije okvir ni programski jezik, Node.js se često koristi za izgradnju pozadinskih usluga, poput Application Programming Interface (API) za internetske ili mobilne aplikacije (Dir.hr, 2024.). Mnoge velike tvrtke koriste Node.js u proizvodnji zbog njegovih prednosti.

Node.js pruža visoke performanse i skalabilnost što ga čini idealnim za izradu brze i visoko skalabilne usluge. Jedna od glavnih prednosti Node.js-a je to što omogućava korištenje JavaScript-a za razvoj kako klijentske, tako i serverske strane aplikacija, što rezultira čistim i

konzistentnim izvornim kodom. Korištenje istog jezika na obje strane olakšava održavanje i konzistenciju koda.

Node.js ima veliki ekosustav knjižnica i modula dostupnih preko npm-a (Node Package Manager), što dodatno ubrzava razvoj. Asinkrona, odnosno ne blokirajuća priroda Node.js-a, omogućava obradu više zahtjeva istovremeno bez blokiranja glavne niti, što je ključna prednost u razvoju modernih aplikacija.

Jedna od glavnih prednosti Node.js-a je njegova jednostavna skalabilnost, koja omogućava jednostavno proširenje aplikacija i dodavanje resursa prema potrebi. Node.js je izvrstan za razvoj internetskih aplikacija u stvarnom vremenu, poput chat aplikacija ili igara, koje zahtijevaju brzu i učinkovitu sinkronizaciju. Njegova arhitektura omogućava brzu obradu asinkronih operacija. Zbog korištenja JavaScript-a, Node.js je jednostavan za učenje i kodiranje, a omogućava i predmemoriranje modula, smanjujući potrebu za ponovnim izvršavanjem koda.

Node.js učinkovito upravlja strujanjem podataka, smanjujući vrijeme učitavanja i omogućavajući brzo strujanje audio i video datoteka. Platforme poput Paas i Heroku omogućavaju jednostavnu implementaciju Node.js aplikacija, što dodatno pojednostavljuje proces razvoja i distribucije. Mnoge poznate tvrtke koriste Node.js, kombinirajući frontend i backend timove u jednu jedinicu, čime se poboljšava suradnja i produktivnost.

Node.js se često koristi za razvoj aplikacija za razgovor u stvarnom vremenu, složenih jednostranih aplikacija, alata za suradnju u stvarnom vremenu, aplikacija za strujanje podataka i aplikacija temeljenih na JSON API-ju. Zbog svoje jednostavnosti, skalabilnosti i mogućnosti brzog izvršavanja koda, Node.js je moćan alat za izgradnju raznih vrsta aplikacija.

1.7. Google Firebase

Google Firebase je platforma za razvoj aplikacija uz podršku Googlea, koja programerima omogućuje stvaranje iOS, Android i web aplikacija. Ova platforma nudi alate za praćenje analitike, izvještavanje i otklanjanje grešaka, kao i za kreiranje marketinških kampanja i eksperimentiranje s proizvodima (Firebase, 2024).

Google Firebase nudi razne usluge, uključujući:

• Google Analytics,

• autentifikaciju,

• slanje poruka u oblaku,

• bazu podataka u stvarnom vremenu,

• Firebase Crashlytics,

• praćenje performansi,

• testni laboratorij.

Google Analytics za Firebase omogućava besplatno i neograničeno izvještavanje za čak pedeset različitih događaja. Ova usluga prikazuje podatke o ponašanju korisnika unutar iOS i Android aplikacija, omogućujući programerima donošenje boljih odluka o unapređenju performansi i marketinškim strategijama.

Firebase Authentication olakšava programerima izradu sigurnih sustava autentifikacije te poboljšava iskustvo prijave i registracije korisnika. Pruža sveobuhvatno rješenje za upravljanje identitetima, uz podršku za prijavu putem e-maila, lozinke, telefonskog broja, te integraciju s Google, GitHub, Facebook, Twitter i drugim računima (Firebase, 2024).

Firebase Cloud Messaging omogućava slanje poruka na više platformi, omogućujući tvrtkama pouzdanu dostavu poruka na iOS, Android i web aplikacije, bez dodatnih troškova.

Firebase Realtime Database je NoSQL baza podataka smještena u oblaku koja omogućava pohranu i sinkronizaciju podataka između korisnika u stvarnom vremenu. Podaci se automatski ažuriraju na svim uređajima, čak i kada aplikacija nije povezana na internet.

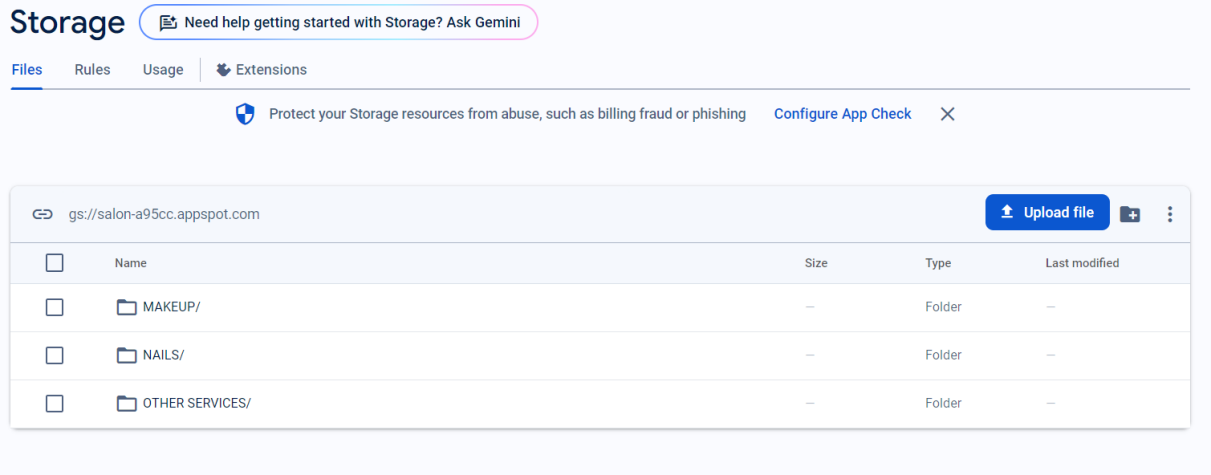
Firebase Crashlytics omogućava programerima praćenje padova aplikacija u stvarnom vremenu, te pomaže u prioritetizaciji i rješavanju problema stabilnosti koji mogu narušiti korisničko iskustvo. Ovo omogućava programerima da više vremena posvete razvoju novih značajki, a manje popravljanju grešaka.

Firebase Performance Monitoring pruža programerima uvid u performanse njihovih iOS i Android aplikacija, pomažući im da identificiraju gdje i kada se performanse mogu poboljšati.

Firebase Test Lab je cloud-based infrastruktura za testiranje aplikacija. Programeri mogu jednostavno testirati svoje iOS i Android aplikacije na različitim uređajima i konfiguracijama, te pregledati rezultate, uključujući videozapise, snimke zaslona i zapisnike, putem Firebase konzole (Firebase, 2024).

1.7.1 Firebase Storage

Firebase Storage je usluga koju pruža Firebase, a omogućuje sigurno pohranjivanje i posluživanje korisničkih datoteka poput slika, videozapisa i drugih vrsta sadržaja. Ova usluga koristi Google Cloud Storage, što osigurava visoku razinu sigurnosti i pouzdanosti podataka. Također se lako integrira s ostalim Firebase uslugama, kao što su Firebase Authentication za kontrolu pristupa i Firestore za pohranu metapodataka, te može skalirati kako bi podržao aplikacije s velikim brojem korisnika.



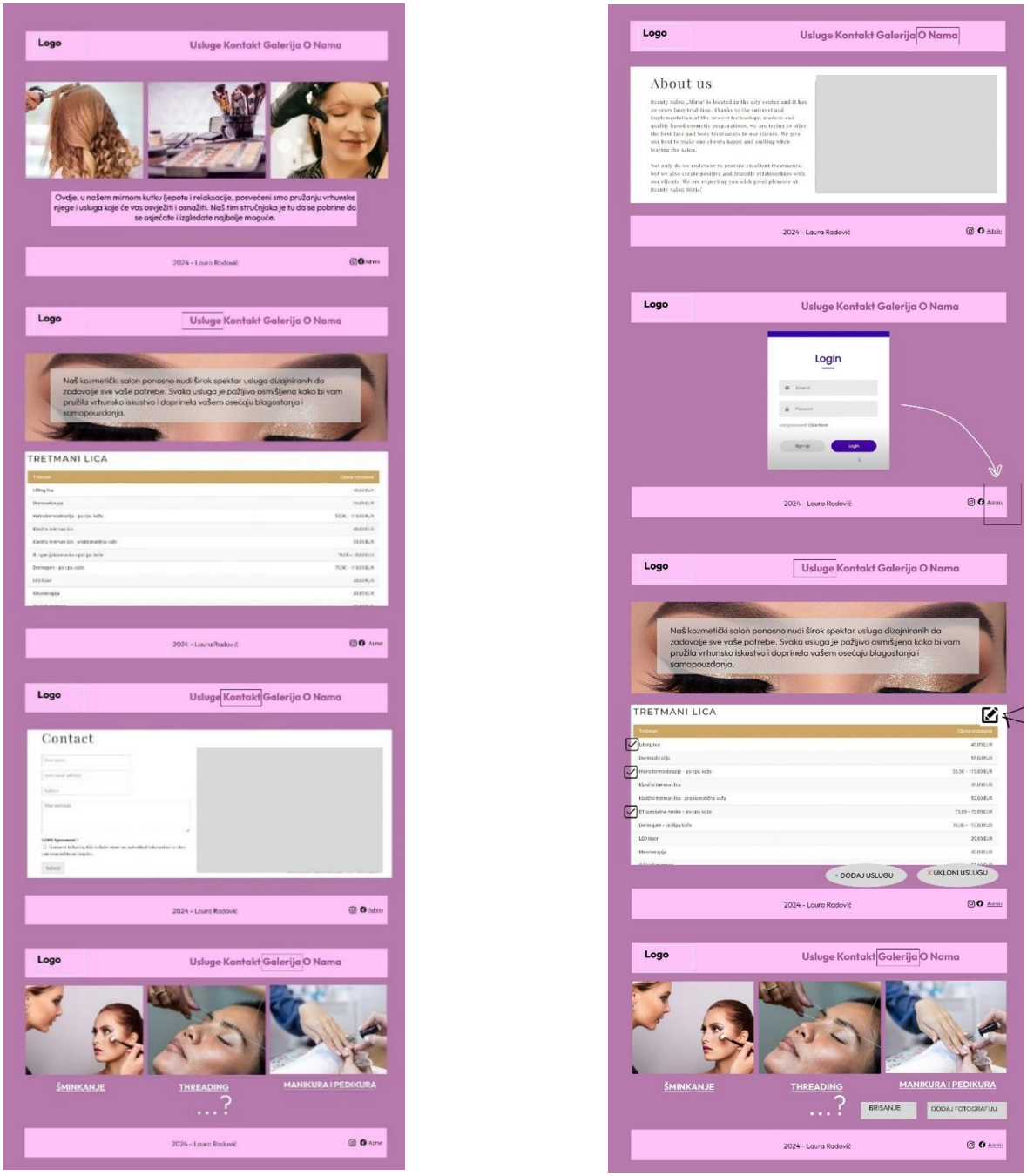
Slika 1. Prikaz kreiranih mapa u Firebase Storage-u

Web aplikacija „Glamour“ koristi tri kreirane mape, gdje je svaka povezana s određenom uslugom koju kozmetički salon nudi. Kreirane mape su prikazane na slici 1. Ova postavka će se implementirati u komponenti galerija gdje će korisnicima biti omogućeno pregledavanje učitanih slika.

2. Prikaz funkcionalnosti aplikacije

2.1. Planiranje izgleda aplikacije

Prije početka izrade web aplikacije i programiranja iste, napravljena je skica u programu *Figma*. Osim boja i odabira fonta, najveće odstupanje od završne verzije aplikacije je vezano uz prikaz opcija dostupnih administratoru. Napravljen je zaseban prikaz administratorske ploče kako bi se pojednostavila sama izrada. Slika 2 prikazuje izrađenu skicu aplikacije.



Slika 2. Izrađena skica u alatu *Figma*

Osim skice dizajna i funkcionalnosti, u aplikaciiji *Figma* je također osmišljen logo inspiriran ljepotom i njegom kojeg prikazuje Slika 3. Sa lijeve strane je prikazan ženski profil, nacrtan tankim linijama koristeći boju #BF8F34. Pored ilustracije nalazi se ispisan naziv kozmetičkog salona, a ispod naziva je sadržana dodatna poruka koja naglašava misiju salon. Za tekst loga se koristio font *Raleway* u crnoj boji #000000.

Slika 3. Prikaz loga

2.2. Instalacija i priprema okruženja

Prije početka razvoja web aplikacije, potrebno je na vlastito računalo instalirati nekoliko alata koji će olakšati i ubrzati cijeli proces. Za izradu ove aplikacije korišten je Visual Studio Code, skraćeno VS Code, koji služi za pisanje i uređivanje koda te pruža podršku za razne razvojne operacije, uključujući otklanjanje grešaka, upravljanje zadacima i kontrolu verzija. Cilj VS Code-a je pružiti razvojnim programerima sve potrebne alate kako bi proces kodiranja, pronalaženja i ispravljanja grešaka bio što brži i jednostavniji.

VS Code je besplatan alat, dostupan za privatnu i komercijalnu upotrebu, što ga čini atraktivnim izborom za programere. Jednostavan je za instalaciju i korištenje, što je bio jedan od glavnih razloga zašto je odabran za ovaj projekt. Nakon instalacije, sljedeći korak je postavljanje Node.js okruženja na računalo kako bi se omogućio pristup npm-u (engl. *Node Package Manager*), koji će se koristiti za preuzimanje svih potrebnih paketa za razvoj aplikacije.

Node.js je otvoreno poslužiteljsko okruženje koje radi na više platformi i koristi JavaScript za backend razvoj. Node.js će poslužiti za prikaz koda u web pregledniku tijekom izrade aplikacije. Nakon uspješne instalacije Node.js-a, potrebno je provjeriti je li ispravno instaliran i može li se koristiti putem terminala, što se može napraviti pokretanjem naredbe „- -*version*“ u naredbenom retku (engl. *Command prompt*).

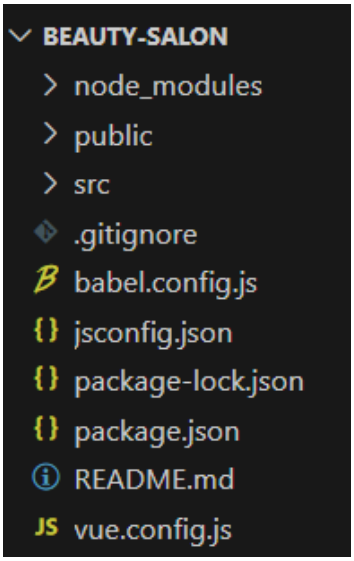
Sljedeći korak u postavljanju radnog okruženja je instalacija Vue.js, JavaScript okvira za izradu korisničkih sučelja. Instalacija Vue.js-a obavlja se putem Vue CLI-a, alata koji omogućava brzo postavljanje novih projekata koristeći jednostavne naredbe u terminalu koja je „npm install -g @vue/cli“. Vue CLI se odnosi na npm paket koji se globalno instalira i pruža mogućnost brzog stvaranja novih projekata uz optimiziranu konfiguraciju za većinu aplikacija.

Prije pokretanja sljedeće naredbe, potrebno se smjestiti unutar željenog direktorija gdje će se izraditi projekt. Naredbi „vue create“ potrebno je dodati naziv direktorija koji će se kreirati te

u kojem će se sve pohranjivati. Nakon kreiranja projekta, potrebno je postaviti osnovne konfiguracije i pokrenuti projekt. Naredba „npm init“ koristi se za kreiranje *package.json* datoteke, koja služi kao glavna datoteka za opisivanje npm paketa.

Za integraciju s Firebase bazom podataka, potrebno je instalirati Firebase paket putem npm-a. Nakon instalacije, Firebase se dodaje u aplikaciju pomoću jednostavne naredbe, omogućujući korištenje svih prednosti koje Firebase nudi.

Cijeli proces razvoja aplikacije odvija se unutar Visual Studio Code-a, gdje je moguće pregledavati i upravljati strukturom projekta, uključujući sve potrebne datoteke i mape. Struktura projekta je prikazana na Slici 4, te uključuje mape poput "node_modules" gdje se nalaze svi instalirani paketi, "public" koja sadrži osnovne postavke aplikacije, i "src" gdje se nalaze sve datoteke koje dinamički pokreću aplikaciju.



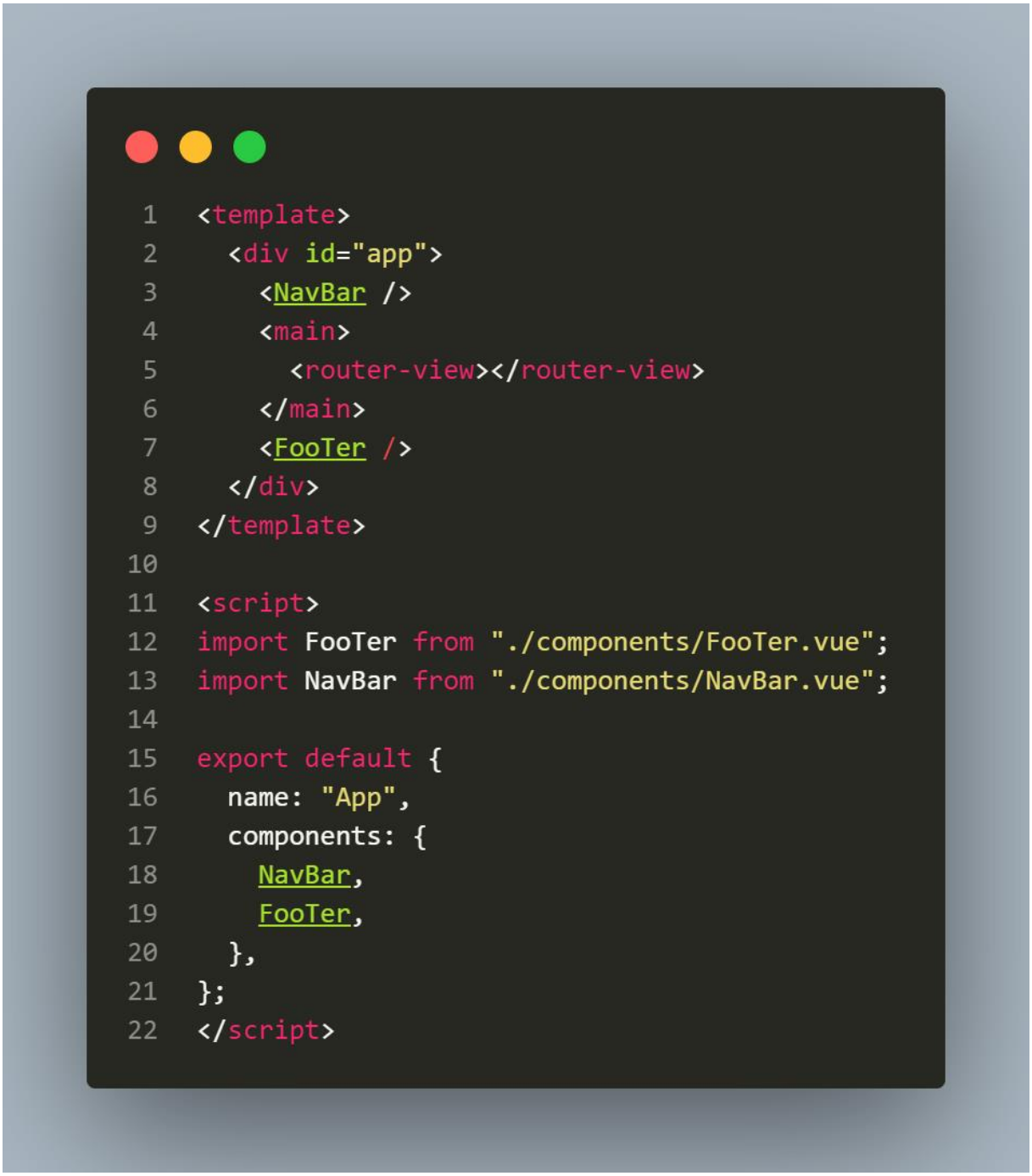
Slika 4. Struktura projekta

Svaka od kreiranih datoteka ima svoju specifičnu funkciju. Datoteka "babel" se koristi za prevođenje novijih verzija JavaScript-a u stariju verziju koja je kompatibilna s većinom preglednika. Uključivanjem datoteke „jsconfig.json“ osigurava se da se sve datoteke tretiraju kao cjelina čime se omogućuje lakša provjera grešaka u projektu. Prilikom pokretanja naredbi poput „npm update/install“, automatski se generira ili ažurira datoteka „package-lock.json“ . Njena je svrha osigurati dosljednost verzija ovisnosti (engl. *dependencies*) u projektu. Drugim riječima, prilikom instaliranja, ažuriranja ili uklanjanja paketa, sve verzije ovisnosti će ostati iste za sve korisnike i okruženja. Datoteka „package.json“ sadrži popis ovisnosti projekta, upravlja njima, te uključuje informacije o projektu kao što su naziv i verzija. Konfiguracijskim postavkama za Vue.js projekt kreiran pomoću Vue CLI upravlja datoteka „vue.config.js“ (Vue CLI Configuration Reference, 2024). Moguće je prilagoditi zadanu konfiguraciju specifičnim potrebama zadanog projekta.

2.3 Struktura i konfiguracija aplikacije

Prije nego što se započne s razvojem web aplikacije, potrebno je pravilno postaviti strukturu projekta i konfigurirati osnovne komponente. U ovom projektu, aplikacija je izrađena koristeći razvojni okvir Vue.js te postoje nekoliko ključnih datoteka na kojima se gradi cijela aplikacije.

Datoteka „*App.vue*“ predstavlja glavnu komponentu aplikacije koja definira osnovnu strukturu aplikacije te je prikazana na slici 5.



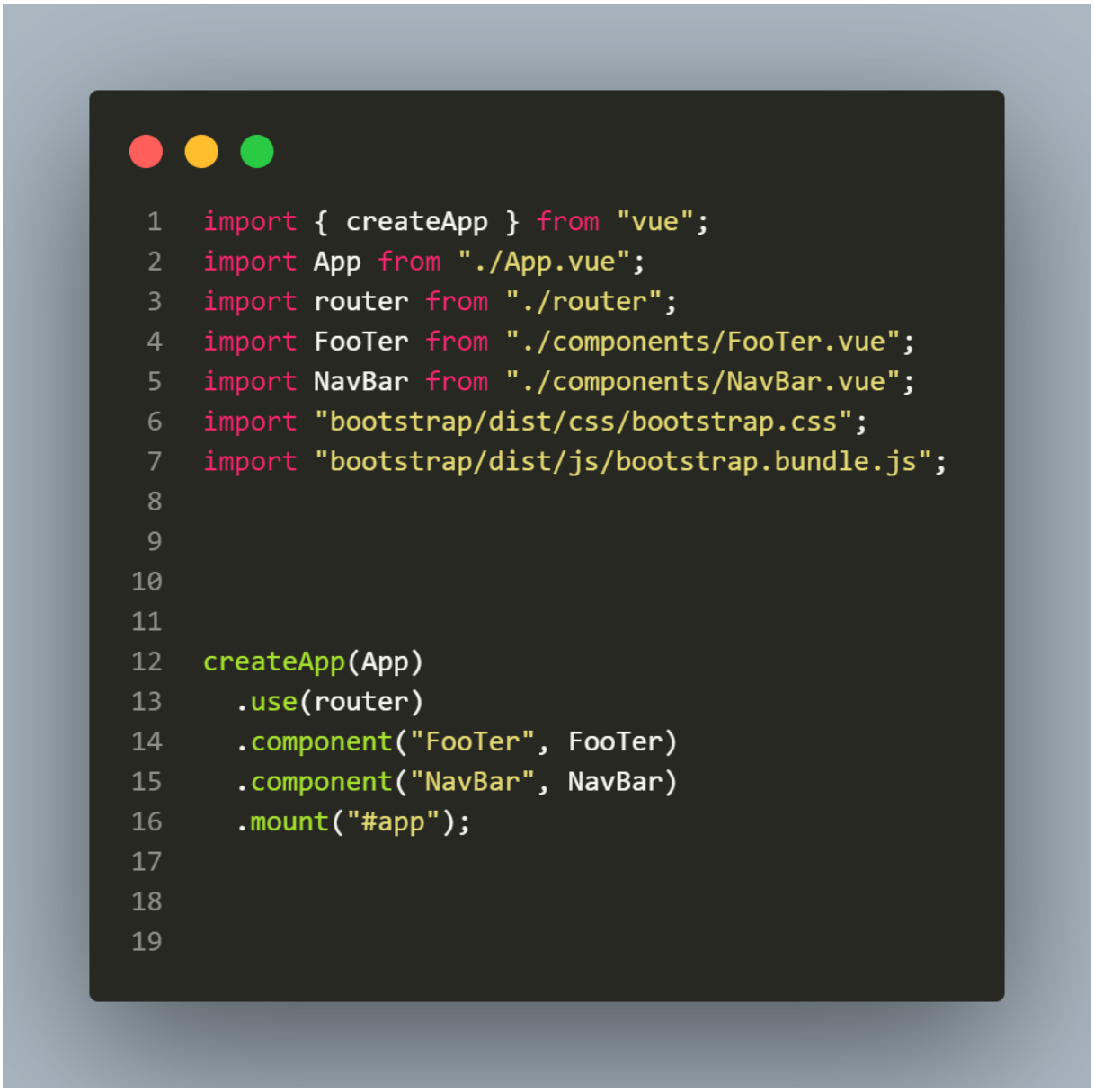
```vue
<template>
  <div id="app">
    <NavBar />
    <main>
      <router-view></router-view>
    </main>
    <FooTer />
  </div>
</template>

<script>
import FooTer from "./components/FooTer.vue";
import NavBar from "./components/NavBar.vue";

export default {
  name: "App",
  components: {
    NavBar,
    FooTer,
  },
};
</script>
```

Slika 5. Prikaz App.vue datoteke

U *template* dijelu se koriste dvije ključne komponente *NavBar* i *FooTer*, koje omogućuju prikaz navigacije i podnožja unutar aplikacije. Ove komponente su uključene u navedenu datoteku kako bi se prikazivale u svim komponentama aplikacije, bez potrebe za njihovim dodatnim uključivanjem u svakoj pojedinačnoj komponenti. Za dinamičko prikazivanje komponenti u zavisnosti od odabrane rute služi „*router-view"* te omogućuje navigaciju kroz različite stranice aplikacije bez ponovnog učitavanja stranice. U skriptnom dijelu se koristi „*import*" za uvoz komponenti koje se nalaze u mapi „*components*". Sintaksa „export default" omogućuje izvoz definirane komponente pod nazivom „App" , čime postaje dostupna drugim dijelovima aplikacije. Uključene komponente „NavBar" i „FooTer" postaju dostupne unutar ove komponente i mogu se koristiti u „template" dijelu. Definirani su također i stilovi koji se primjenjuju na cijelu aplikaciju. Na primjer, koristi se Google font *Roboto* koji je uvezen pomoću direktive „@import", te su definirani responzivni stilovi pomoću *media query-a* koji prilagođavaju izgled aplikacije za različite veličine ekrana.

Datoteka „*main.js"* je ulazna točka aplikacije. U njoj se integrira Bootstrap, CSS i JavaScript bundle, što omogućava korištenje gotovih stilova i komponenti za brži razvoj. Uz navedeno, uvozi se glavna komponenta aplikacije iz datoteke „*App.vue*" gdje je sadržana struktura i konfigurira se *Vue Router* za upravljanje navigacijom između različitih stranica aplikacije. Kreira se instanca aplikacije pomoću Vue API-a „*createApp"*, dok se metodom *„mount"* aplikacija montira na HTML element s „*id=app"* čime postaje vidljiva i funkcionalna u pregledniku. Sadržaj datoteke je prikazan na slici 6.

Unutar mape „*router*" nalazi se datoteka „*index.js*" gdje su definirane rute koje aplikacija koristi za navigaciju. Na početku se učitavaju sve komponente, kao i funkcije „*createRouter*" te „*createWebHistory*" koje omogućuju kreiranje *routera*. Zatim se definira konstanta „*routes*" kao niz objekata, gdje svaki objekt predstavlja jednu rutu koja se sastoji od putanje, naziva i komponente. Na primjer, kada korisnik posjeti „/about", prikazat će se komponenta „AboutPage". Datoteka „*index.js*" je prikazana na slici 7.



```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import FooTer from "./components/FooTer.vue";
import NavBar from "./components/NavBar.vue";
import "bootstrap/dist/css/bootstrap.css";
import "bootstrap/dist/js/bootstrap.bundle.js";




createApp(App)
  .use(router)
  .component("FooTer", FooTer)
  .component("NavBar", NavBar)
  .mount("#app");
```

Slika 6. Prikaz main.js datoteke

```
import { createRouter, createWebHistory } from "vue-router";

import AboutPage from "../views/AboutPage.vue";
import ContactPage from "../views/ContactPage.vue";
import GalleryPage from "../views/GalleryPage.vue";
import ServicesList from "../views/ServicesList.vue";
import HomePage from "@/views/HomePage.vue";
import AdminLogin from "@/views/AdminLogin.vue";
import AdminPage from "@/views/AdminPage.vue";

const routes = [
  {
    path: "/",
    name: "Home",
    component: HomePage,
  },
  {
    path: "/about",
    name: "About",
    component: AboutPage,
  },
  {
    path: "/contact",
    name: "Contact",
    component: ContactPage,
  },
  {
    path: "/gallery",
    name: "Gallery",
    component: GalleryPage,
  },
  {
    path: "/services",
    name: "Services",
    component: ServicesList,
  },
  {
    path: "/login",
    name: "Login",
    component: AdminLogin,
  },
  {
    path: "/admin",
    name: "Admin",
    component: AdminPage,
  },
];

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
});

export default router;

```

Slika 7. Prikaz router konfiguracije

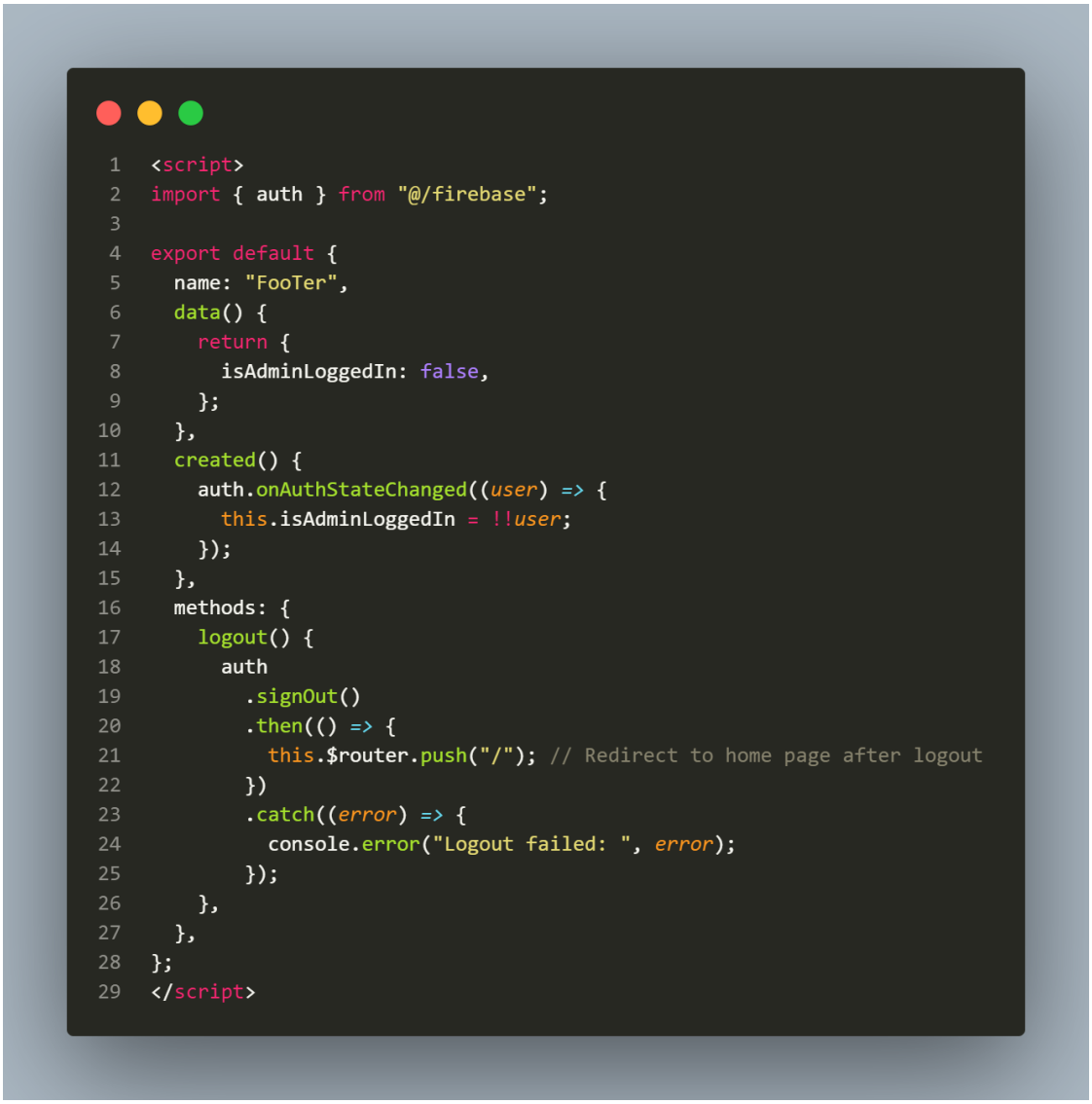
2.4 Implementacija glavnih komponenti korisničkog sučelja

Za učinkovito upravljanje prikazom korisničkog sučelja i osiguravanje dosljednog izgleda na svim stranicama, korištene su dvije glavne komponente: *NavBar* i *FooTer*. Ove komponente su ključne za navigaciju kroz aplikaciju i prikaz osnovnih informacija na svakoj stranici.

Komponenta „Footer“ predstavlja podnožje aplikacije te se iz tog razloga koristi HTML tag *footer* u *template* dijelu koji je prikazan na slici 8. Unutar podnožja nalaze se osnovne informacije o autorskim pravima, kao i veze do društvenih mreža i administrativnog dijela aplikacije. Uključena je ikona za mrežu Instagram te je postavljena kao link kako bi se korisnicima omogućio brz pristup profilu aplikacije. Direktiva „v-if“ se koristi kako bi se prikazale različite opcije korisnicima, ovisno o tome je li prijavljeni korisnik administrator. Ukoliko je to slučaj, prikazuje se veza za pristup administrativnoj stranici kao i gumb za odjavu.

```
<template>
  <footer>
    <p>&copy; 2024 Laura Radović. All rights reserved.</p>
    <div class="right-section">
      <a href="https://www.instagram.com/fidit_uniri/"
        ><img src="@/assets/instagram-icon.png" alt="Instagram"
      /></a>
      <router-link v-if="isAdminLoggedIn" to="/admin" class="admin-link"
        >Admin</router-link
      >
      <router-link v-else to="/login" class="admin-link">Login</router-link>
      <a v-if="isAdminLoggedIn" href="#" class="admin-link" @click="logout"
        >Logout</a
      >
    </div>
  </footer>
</template>
```

Slika 8. Template dio *footer* datoteke



```
<script>
import { auth } from "@/firebase";

export default {
  name: "FooTer",
  data() {
    return {
      isAdminLoggedIn: false,
    };
  },
  created() {
    auth.onAuthStateChanged((user) => {
      this.isAdminLoggedIn = !!user;
    });
  },
  methods: {
    logout() {
      auth
        .signOut()
        .then(() => {
          this.$router.push("/"); // Redirect to home page after logout
        })
        .catch((error) => {
          console.error("Logout failed: ", error);
        });
    },
  },
};
</script>
```

Slika 9. Skriptni dio *footer* datoteke

Slika 9 prikazuje skriptni dio gdje se definira logika autentifikacije korisnika. Komponenta koristi Firebase autentifikaciju za upravljanje prijavom i odjavom korisnika. Pomoću metode „auth.onAuthStateChanged“ provjerava se promjena stanja prijave korisnika, odnosno određuje je li korisnik prijavljen kao administrator. Kada se administrator odluči odjaviti, u metodi „logout“ se poziva funkcija „auth.signOut“ koja će nakon uspješnog izvršavanja preusmjeriti korisnika na početnu stranicu aplikacije. U *style* sekciji, stilovi su prilagođeni kako bi podnožje ostalo fiksirano na dnu ekrana bez obzira na sadržaj stranice, te se osigurava konzistentan izgled sa svim elementima pravilno poravnatim.

```
<template>
  <div>
    <header>
      <div class="logo">
        <img src="@/assets/logo.png" alt="Logo" />
      </div>
      <nav>
        <ul>
          <li><router-link to="/">Home</router-link></li>
          <li><router-link to="/about">O nama</router-link></li>
          <li><router-link to="/contact">Kontakt</router-link></li>
          <li><router-link to="/gallery">Galerija</router-link></li>
          <li><router-link to="/services">Usluge</router-link></li>
        </ul>
      </nav>
    </header>
  </div>
</template>
```

Slika 10. Prikaz navbar komponente

Komponenta *NavBar* odgovorna je za prikaz navigacijske trake koja korisnicima omogućava pristup glavnim sekcijama aplikacije kao što su početna stranica, informacije o uslugama, galerija, kontakt stranica i informacije o salonu. Slika 10 prikazuje *template* dio komponente gdje je uključen logo te lista sa stavkama, element „<ul>“, u kojoj se koriste „router-link“ elementi za navigaciju do različitih ruta unutar aplikacije. Stilizacija navigacijske trake uključuje jednostavne, čiste linije i korištenje boja koje odgovaraju cjelokupnom dizajnu aplikacije.

2.5 Views

U kontekstu Vue.js, "views" su komponente koje predstavljaju glavne stranice ili odjeljke unutar aplikacije. Ove komponente su ključne za navigaciju unutar aplikacije, omogućujući korisnicima prebacivanje između različitih dijelova, bilo da se radi o Single Page Application (SPA) ili više-straničnoj arhitekturi.

Svaki „view“ u Vue.js aplikaciji obično odgovara jednoj URL ruti i sadrži poslovnu logiku specifičnu za tu stranicu. Ove komponente su povezane s Vue Routerom, što omogućava korisnicima navigaciju između različitih ruta bez potrebe za ponovnim učitavanjem stranice.

Aplikacija sadrži sljedeće poglede:

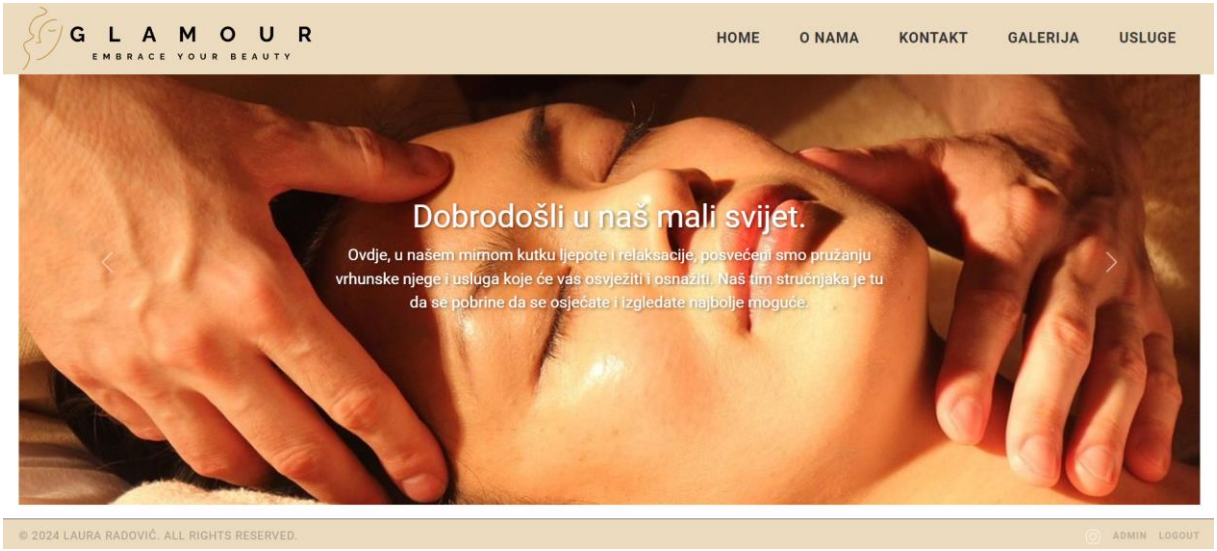
- **HomePage.vue**: Početna stranica aplikacije, koja sadrži osnovne informacije i prikazuje različite funkcionalnosti dostupne na stranici.
- **AboutPage.vue**: Stranica koja pruža informacije o web stranici ili kompaniji, uključujući misiju, viziju i tim.
- **ContactPage.vue**: Kontakt forma koja omogućava korisnicima slanje upita vlasnicima stranice putem emaila, uz integraciju Google karte koja prikazuje lokaciju.
- **GalleryPage.vue**: Stranica sa galerijom slika, najvjerojatnije prikazuje fotografije salona i usluga koje se nude.
- **ServicePage.vue**: Stranica koja prikazuje informacije o različitim uslugama koje pruža salon, omogućujući korisnicima da saznaju više o ponudama.

U nastavku će biti detaljno analizirane funkcionalnosti i implementacije svakog od pregleda.

2.5.1 HomePage.vue

Početna stranica (*HomePage.vue*) predstavlja ulaznu točku aplikacije. Sadrži ključne informacije i služi kao središnje mjesto za navigaciju ka drugim stranicama. Slika 11 prikazuje kako izgleda početna stranica aplikacije "Glamour" u pregledniku.

U *template* dijelu, prikazanom na slici 12, se koristi *hero* sekcija, koja uključuje karusel za prikazivanje tri slike vezane uz usluge koje kozmetički salon nudi. Karusel omogućava automatsko ili ručno prebacivanje između slika. Osim navedenoga, sadržan je i tekst koji se prikazuje iznad slika gdje je opisana misija salona.



Slika 11. Prikaz početne stranice web aplikacije

```
<template>
  <div class="main-content">
    <section class="hero">
      <div id="carousel" class="carousel slide" data-bs-ride="carousel">
        <div class="carousel-inner">
          <div class="carousel-item active">
            <img
              src="@/assets/slide1.jpg"
              class="d-block w-100"
              alt="Slide 1"
            />
          </div>
          <div class="carousel-item">
            <img
              src="@/assets/slide2.jpg"
              class="d-block w-100"
              alt="Slide 2"
            />
          </div>
          <div class="carousel-item">
            <img
              src="@/assets/slide3.jpg"
              class="d-block w-100"
              alt="Slide 3"
            />
          </div>
        </div>
        <button
          class="carousel-control-prev"
          type="button"
          data-bs-target="#carousel"
          data-bs-slide="prev"
        >
          <span class="carousel-control-prev-icon" aria-hidden="true"></span>
          <span class="visually-hidden">Prethodno</span>
        </button>
        <button
          class="carousel-control-next"
          type="button"
          data-bs-target="#carousel"
          data-bs-slide="next"
        >
          <span class="carousel-control-next-icon" aria-hidden="true"></span>
          <span class="visually-hidden">Sljedeće</span>
        </button>
      </div>

      <div class="hero-text">
        <h1>Dobrodošli u naš mali svijet.</h1>
        <p>
          Ovdje, u našem mirnom kutku ljepote i relaksacije, posvećeni smo
          pružanju vrhunske njege i usluga koje će vas osvježiti i osnažiti. Naš
          tim stručnjaka je tu da se pobrine da se osjećate i izgledate najbolje
          moguće.
        </p>
      </div>
    </section>
    <section class="services"></section>
  </div>
</template>
```

Slika 12. Prikaz koda početne stranice

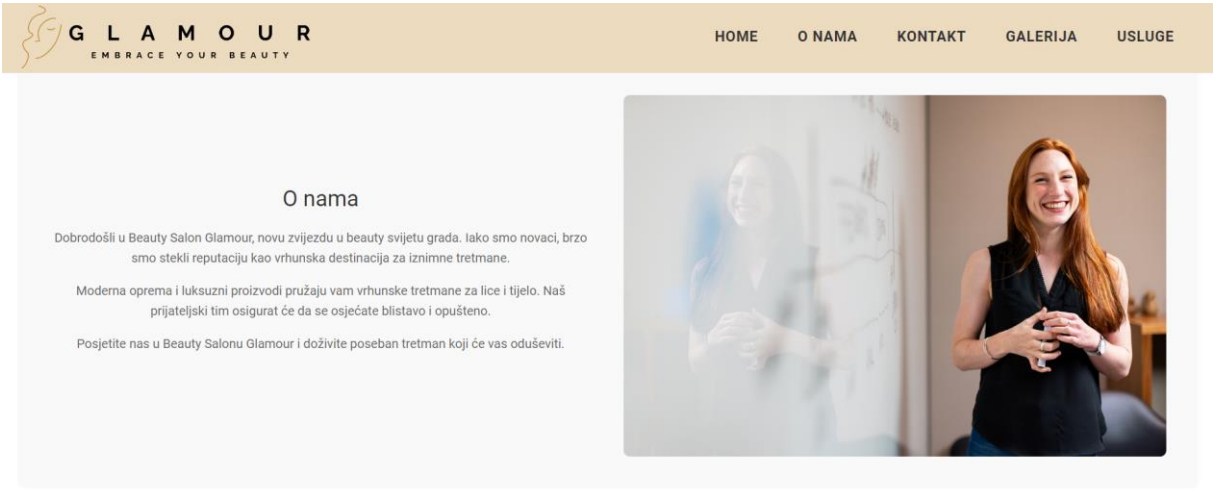
2.5.2 AboutPage.vue

Komponenta *AboutPage.vue* služi za prikazivanje informacija o poslovanju. Sadržava tekstualne i vizualne elemente koji objašnjavaju misiju, viziju, povijest salona te naglašavaju posvećenost tima pružanju visokokvalitetnih usluga. Slika 13 prikazuje istoimenu datoteku.

```vue
<template>
  <div class="about-component">
    <div class="about-content">
      <h1>O nama</h1>
      <p>
        Dobrodošli u Beauty Salon Glamour, novu zvijezdu u beauty svijetu grada.
        Iako smo novaci, brzo smo stekli reputaciju kao vrhunska destinacija za
        iznimne tretmane.
      </p>
      <p>
        Moderna oprema i luksuzni proizvodi pružaju vam vrhunske tretmane za
        lice i tijelo. Naš prijateljski tim osigurat će da se osjećate blistavo
        i opušteno.
      </p>
      <p>
        Posjetite nas u Beauty Salonu Glamour i doživite poseban tretman koji će
        vas oduševiti.
      </p>
    </div>
    <div class="about-image">
      <img src="@/assets/aboutUs.jpg" alt="About Us" />
    </div>
  </div>
</template>

<script>
export default {
  name: "AboutPage",
};
</script>
```

Slika 13. Prikaz komponente "O nama"

Slika 14 prikazuje komponentu "O nama“ u pregledniku.



Slika 14. Prikaz stranice "O nama"

2.5.3 ContactPage.vue

Kontakt stranica (*ContactPage.vue*) omogućava korisnicima da stupe u kontakt putem email forme. Također, prikazuje Google mapu s lokacijom salona. Na slici 15 prikazana je implementacije forme koja se prikazuje korisniku.

```
<form @submit.prevent="submitForm">
          <div class="form-group">
            <label for="name">Vaše ime</label>
            <input
              type="text"
              id="name"
              name="name"
              class="form-control"
              v-model="form.name"
              required
            />
          </div>
          <div class="form-group">
            <label for="email">Vaša e-mail adresa</label>
            <input
              type="email"
              id="email"
              name="email"
              class="form-control"
              v-model="form.email"
              required
            />
          </div>
          <div class="form-group">
            <label for="date">Datum</label>
            <input
              type="date"
              id="date"
              name="date"
              class="form-control"
              v-model="form.date"
              required
            />
          </div>
          <div class="form-group">
            <label for="time">Vrijeme</label>
            <select
              id="time"
              name="time"
              class="form-control"
              v-model="form.time"
              required
            >
              <option value="12:00">12:00</option>
              <option value="13:00">13:00</option>
              <option value="14:00">14:00</option>
              <option value="15:00">15:00</option>
              <option value="16:00">16:00</option>
              <option value="17:00">17:00</option>
              <option value="18:00">18:00</option>
              <option value="19:00">19:00</option>
              <option value="20:00">20:00</option>
            </select>
          </div>
          <div class="form-group">
            <label for="message">Molimo navesti usluge koje želite:</label>
            <textarea
              id="message"
              name="message"
              class="form-control"
              rows="5"
              v-model="form.message"
              required
            ></textarea>
          </div>
          <div class="form-group form-check">
            <input
              type="checkbox"
              class="form-check-input"
              id="gdprConsent"
              v-model="form.gdprConsent"
              required
            />
            <label class="form-check-label" for="gdprConsent">
              Slažem se da ovo web mjesto pohranjuje moje poslane podatke na
              takav način da mogu odgovoriti na moj upit.
            </label>
          </div>
          <button type="submit" class="btn btn-primary btn-block">
            Pošalji
          </button>
        </form>
```

Slika 15. Prikaz dijela koda kontakt forme

Svaka „form-group“ sadrži odgovarajuće polje poput unosa imena, email adrese, datuma, vremena (s odabirom unaprijed definiranih intervala“ te tekstualno polje za navođenje željenih usluga. Dodan je i obavezni *checkbox* za pristanak korisnika na pohranu i obradu podataka. Klikom na gumb „Pošalji“, forma pokreće metodu „submitForm“.

2.5.3.1 EmailJS integracija:

EmailJS je servis koji omogućava slanje emailova direktno iz JavaScript aplikacije, bez potrebe za vlastitim email serverom ili *backend* infrastrukturom. Ovaj servis se često koristi u razvoju web aplikacija zbog svoje jednostavnosti i fleksibilnosti. Njegova implementacija je prikazana na slici 16.

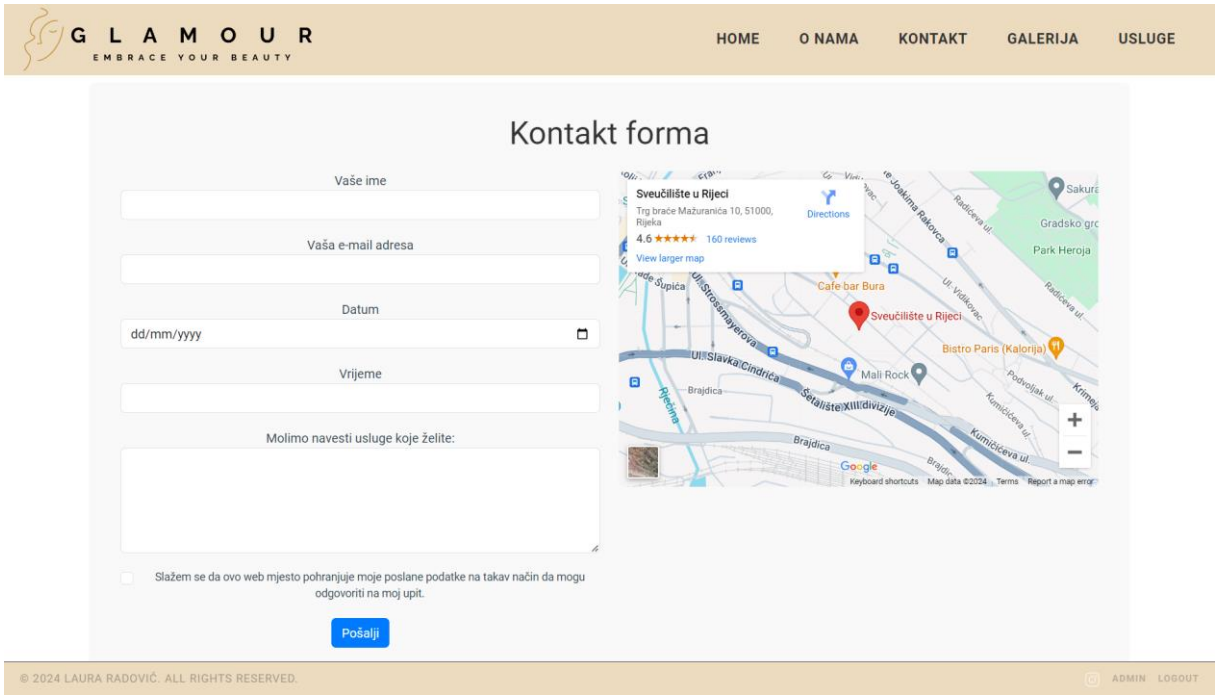
```
<script>
import emailjs from "emailjs-com";

export default {
  name: "ContactPage",
  data() {
    return {
      form: {
        name: "",
        email: "",
        date: "",
        time: "",
        message: "",
        gdprConsent: false,
      },
    };
  },
  methods: {
    submitForm() {
      if (!this.form.message.trim()) {
        alert("Molimo navedite usluge koje želite.");
        return;
      }

      const templateParams = { ...this.form };

      emailjs
        .send(
          "service_4395q0h",
          "template_j8wb3qv",
          templateParams,
          "oF-77kDqtLD-u5pV1"
        )
        .then(
          (response) => {
            console.log("SUCCESS!", response.status, response.text);

            var successModal = new window.bootstrap.Modal(
              document.getElementById("successModal")
            );
            successModal.show();

            this.form = {
              name: "",
              email: "",
              date: "",
              time: "",
              message: "",
              gdprConsent: false,
            };
          },
          (error) => {
            console.log("FAILED...", error);
          }
        );
    },
  },
};
</script>
```

Slika 16. Skriptni dio kontakt komponente

U web aplikaciji za kozmetički salon, EmailJS je integriran u aplikaciju kako bi omogućio slanje upita direktno putem kreirane kontakt forme. U skriptnom dijelu uključena je metoda „*send*“ koja prima ID usluge, predloška i korisnički ID, čime se email šalje izravno iz korisničkog sučelja. U slučaju uspješnog slanja, korisniku se prikazuje modal s potvrdom, dok se greške evidentiraju u konzoli.

Korištenje EmailJS-a značajno je pojednostavilo razvoj aplikacije jer nije bilo potrebe za postavljanjem vlastitog email servera, a integracija u aplikaciju bila je brza i jednostavna.

Na slici 17. prikazan je izgled „Kontakt“ stranice u pregledniku.



Slika 17. Prikaz stranice kontakt forme

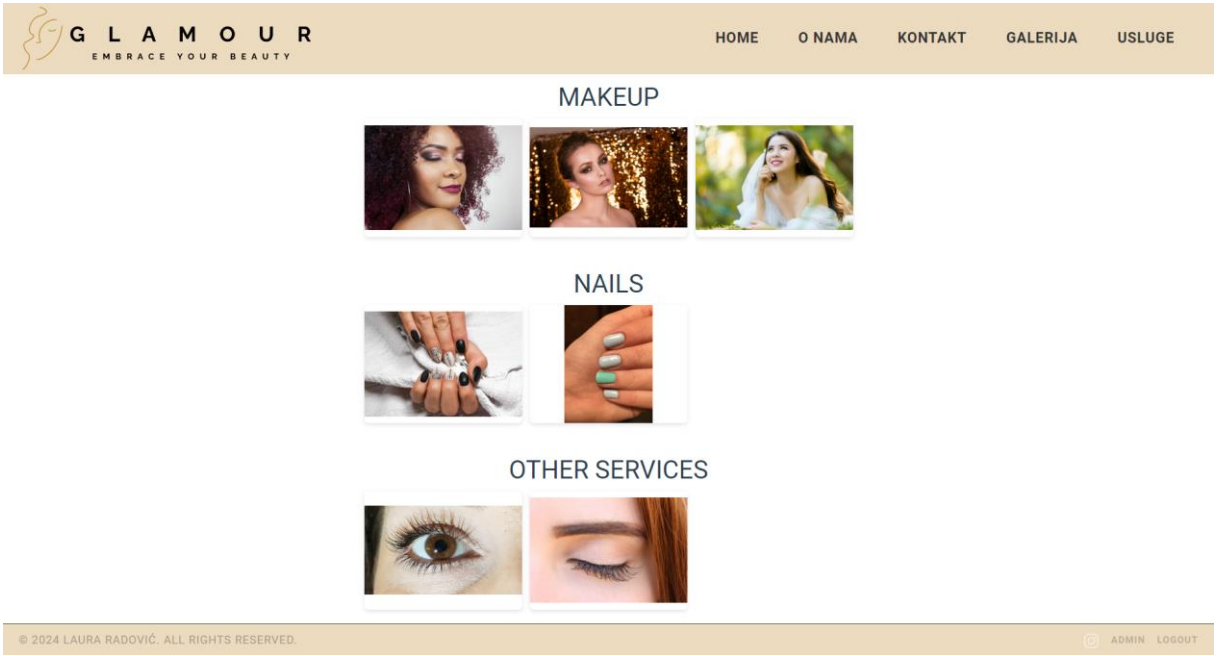
.

2.5.4 GalleryPage.vue

Za prikazivanje galerije slika koristi se komponenta *GalleryPage.vue*, koja uključuje fotografije salona, klijenata i usluga. Na slici 18. je prikazan skriptni dio koda komponente, a na slici 19. je njen prikaz u pregledniku. Ova komponenta omogućava dinamičko dohvaćanje slika iz Firebase Storage-a i grupiranje po mapama. U metodi „created“ poziva se „fetchFoldersAndPhotos“ koja dohvaća sve mape koristeći funkciju „listAll“. Zatim se za svaki folder, pomoću metode „fetchPhotosFromFolder“ dohvaćaju sve slike unutar mape, generirajući URL-ove. Spremaju se pod „groupedPhotos“ koji omogućava grupiranje slika prema imenima foldera.

```vue
<script>
import { storage } from "@/firebase";
import { ref, listAll, getDownloadURL } from "firebase/storage";

export default {
  name: "GalleryView",
  data() {
    return {
      groupedPhotos: {},
    };
  },
  created() {
    this.fetchFoldersAndPhotos();
  },
  methods: {
    async fetchFoldersAndPhotos() {
      const rootRef = ref(storage);
      try {
        const res = await listAll(rootRef);
        for (const folderRef of res.prefixes) {
          const folderName = folderRef.name;
          await this.fetchPhotosFromFolder(folderRef, folderName);
        }
      } catch (error) {
        console.error("Error fetching folders and photos: ", error);
      }
    },
    async fetchPhotosFromFolder(folderRef, folderName) {
      try {
        const res = await listAll(folderRef);
        const photoUrls = await Promise.all(
          res.items.map(async (itemRef) => {
            const url = await getDownloadURL(itemRef);
            return { url };
          })
        );

        this.groupedPhotos[folderName] = photoUrls;
      } catch (error) {
        console.error("Error fetching photos from folder:", folderName, error);
      }
    },
  },
};
</script>
```

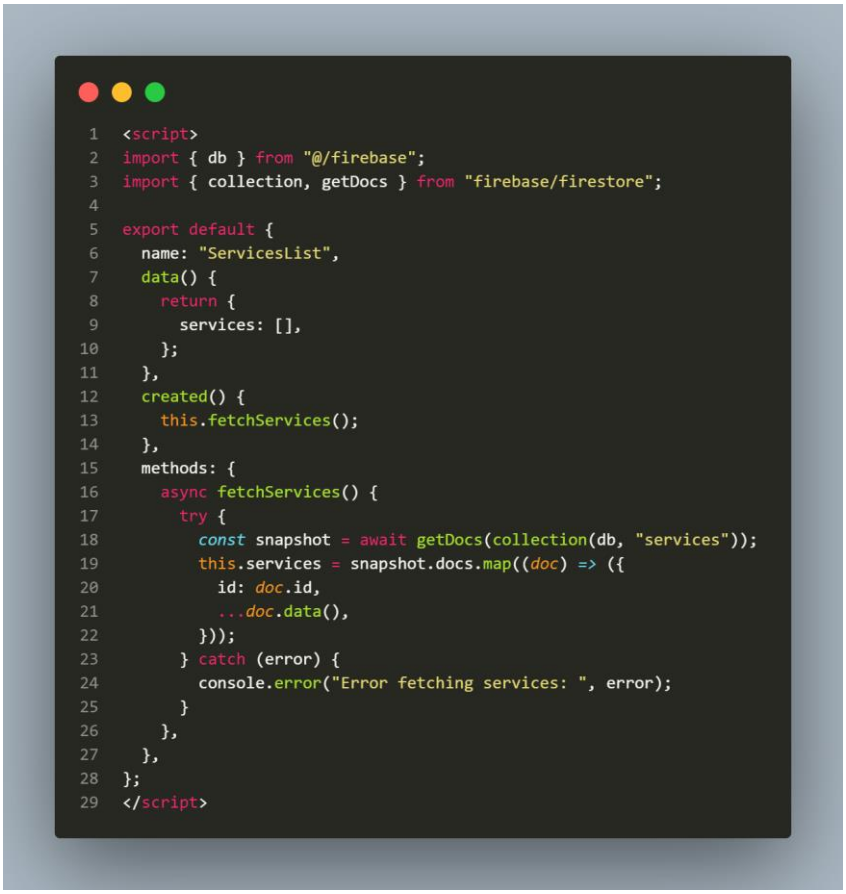
Slika 18. Prikaz skriptnog dijela komponente za galeriju



Slika 19. Prikaz stranice galerije

2.5.5 ServicePage.vue

Komponenta *ServicePage.vue* prikazuje detalje o uslugama koje salon pruža, kao što su frizerske usluge, manikura, pedikura. Ova stranica omogućava korisnicima da se upoznaju s ponudom i potencijalno zakažu termin.



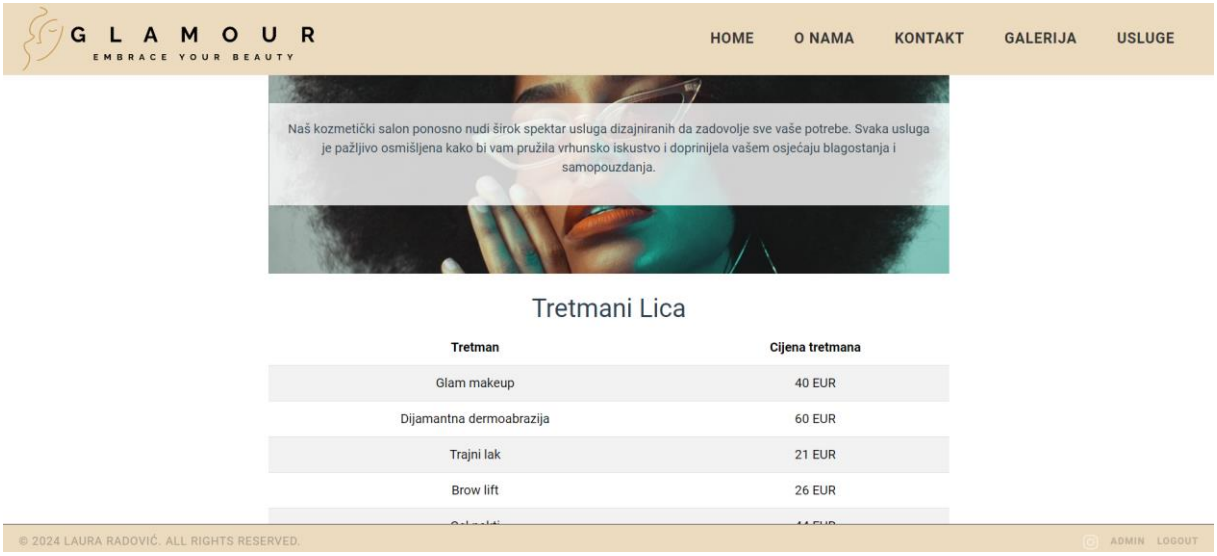
```vue
<script>
import { db } from "@/firebase";
import { collection, getDocs } from "firebase/firestore";

export default {
  name: "ServicesList",
  data() {
    return {
      services: [],
    };
  },
  created() {
    this.fetchServices();
  },
  methods: {
    async fetchServices() {
      try {
        const snapshot = await getDocs(collection(db, "services"));
        this.services = snapshot.docs.map((doc) => ({
          id: doc.id,
          ...doc.data(),
        }));
      } catch (error) {
        console.error("Error fetching services: ", error);
      }
    },
  },
};
</script>
```

Slika 20. Prikaz skriptnog dijela koda stranice usluga

Slika 20. prikazuje skriptni dio koda komponente koja koristi Firebase Firestore za dohvaćanje podataka iz kolekcije imena „services“. Metoda „fecthServices“ asinkrono dohvaća dokumente iz baze podataka koristeći „getDocs“ i „collection“.

Usluge su prikazane u listi, s osnovnim informacijama kao što su ime i cijena, što korisnicima omogućava brz pregled ponude.

Slika 21. prikazuje komponentu „Usluge“ u pregledniku.



Slika 21. Prikaz stranice "Usluge"

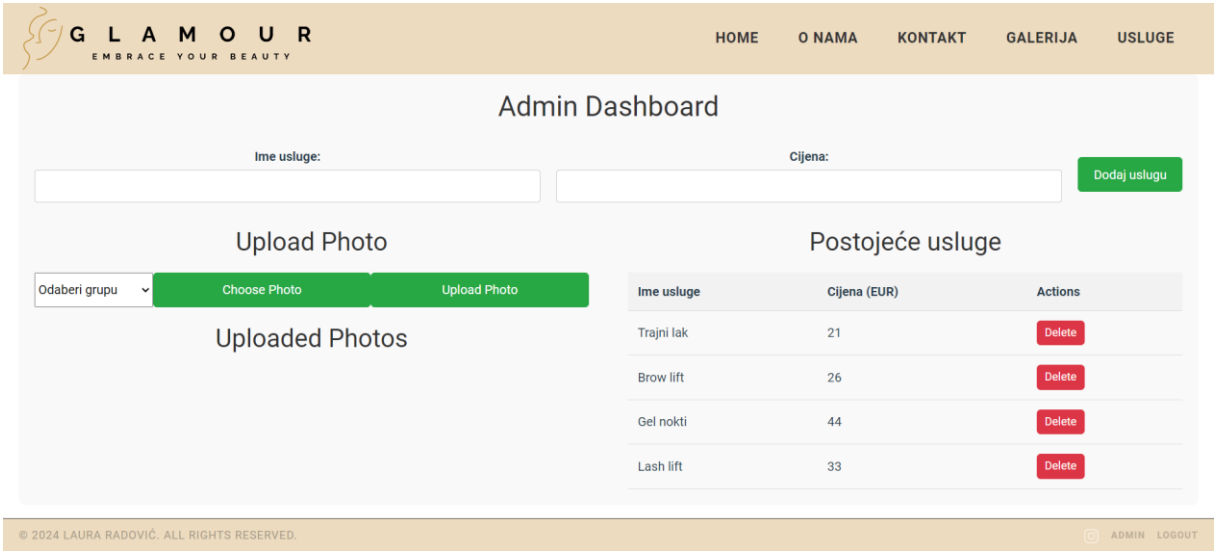
2.6. Autentikacija i Admin Dashboard

U „*AdminLogin.vue*“ datoteci koristi se Firebase Authentication za autentifikaciju administratora putem email adrese i lozinke. Implementacija je prikazana na slici 21. Kada korisnik klikne gumb za prijavu, poziva se metoda „login“, koja koristi funkciju od Firebase-a „signInWithEmailAndPassword“ čime se provjeravaju uneseni podaci.

```
<script>
import { getAuth, signInWithEmailAndPassword } from "firebase/auth";

export default {
  data() {
    return {
      email: "",
      password: "",
    };
  },
  methods: {
    async login() {
      try {
        const auth = getAuth();
        await signInWithEmailAndPassword(auth, this.email, this.password);
        this.$router.push("/"); // Redirect to admin dashboard after login
      } catch (error) {
        alert(error.message);
      }
    },
  },
};
</script>
```

Slika 22.Prikaz skriptnog dijela koda komponente prijave

Ako su podaci ispravni, korisnika se preusmjerava na administrativnu ploču pomoću *routera*, dok se u slučaju pogreške prikazuje poruka o pogrešci. Slika 22 prikazuje stranicu gdje administrator može upravljati uslugama i fotografijama.



Slika 23. Prikaz stranice administratora

2.7. Funkcionalnosti za dodavanje i brisanje usluga

U komponenti *AdminPage.vue*, implementirane su funkcionalnosti za upravljanje uslugama i fotografijama. Ova komponenta omogućuje administratoru dodavanje novih usluga, prikaz postojećih usluga, te brisanje usluga iz baze podataka. Sve ove funkcionalnosti oslanjaju se na Firebase Firestore bazu podataka.

```
async addService() {
    try {
      const newServiceRef = await addDoc(
        collection(db, "services"),
        this.newService
      );
      this.services.push({
        id: newServiceRef.id,
        ...this.newService,
      });
      this.newService.name = "";
      this.newService.price = null;
    } catch (error) {
      console.error("Error adding service: ", error);
    }
```

Slika 24. Prikaz dijela koda za dodavanje usluga

Na slici je prikazana metoda „addService“ koja služi za dodavanje nove usluge u Firestore bazu podataka koristeći funkciju „addDoc“ iz Firebase Firestore biblioteke. Prvo se stvara referenca na novi dokument u kolekciji „services“ i spremaju podaci o usluzi. Nakon uspješnog dodavanja, nova usluga se dodaje u lokalni popis usluga „this.services“ i osvježava korisničko sučelje, a polja „name“ i „price“ se resetiraju. U slučaju pogreške tijekom dodavanja, prikazuje se poruka o grešci.

2.8. Brisanje usluga

Brisanje usluga implementirano je metodom „deleteService“, koja prihvaća ID usluge koja treba biti obrisana i uklanja je iz Firestore baze podataka koristeći funkciju „deleteDoc“. Nakon toga se ažurira popis prikazanih usluga filtriranjem postojećeg popisa „this.services“ i uklanjanjem usluge s obrisanim ID-em. Metoda je prikazana na slici 24.

```
async deleteService(id) {
      try {
        await deleteDoc(doc(db, "services", id));
        this.services = this.services.filter((service) => service.id !== id);
      } catch (error) {
        console.error("Error deleting service: ", error);
      }
```

Slika 25. Prikaz dijela koda za brisanje usluga

2.9. Funkcionalnosti za upravljanje fotografijama

Osim upravljanja uslugama, komponenta također omogućuje upravljanje fotografijama što je prikazano na slici 25. Metoda „uploadPhoto“ omogućava učitavanje novih fotografija u odabrani folder unutar Firebase Storage-a, koristeći funkcije „ref“ i „uploadBytes“. Nakon uspješnog učitavanja, ažuriraju se lokalne varijable i poziva se „fetchPhotos“ za osvježavanje korisničkog prikaza. Metoda „fetchFolders“ dohvaća popis dostupnih mapa iz Storage-a koristeći „listAll“. Metoda „fetchFolders“ je odgovorna za dohvaćanje fotografija iz odabranog foldera, također koristeći „listAll“ za dobivanje popisa stavki, a zatim dohvaća URL-ove za preuzimanje svake fotografije.

Ovaj kod ilustrira kako koristiti Firebase Storage za upravljanje datotekama u Vue.js aplikaciji. Zajedno s drugim komponentama, pruža kompletno rješenje za administraciju usluga i medijskih datoteka u jednostavnom i korisnički prihvatljivom sučelju.

```
async uploadPhoto() {
      if (this.selectedFile && this.selectedFolder) {
        try {
          const storageRef = ref(
            storage,
            `${this.selectedFolder}/${this.selectedFile.name}`
          );
          await uploadBytes(storageRef, this.selectedFile);
          this.selectedFile = null;
          this.previewImage = null;
          this.fetchPhotos();
          console.log("Photo uploaded successfully.");
        } catch (error) {
          console.error("Error uploading photo: ", error);
        }
      }
    },
    async fetchFolders() {
      try {
        const listRef = ref(storage);
        const res = await listAll(listRef);
        this.folders = res.prefixes.map((folderRef) => folderRef.name);
      } catch (error) {
        console.error("Error fetching folders: ", error);
      }
    },
    async fetchPhotos() {
      try {
        if (!this.selectedFolder) return;
        const listRef = ref(storage, this.selectedFolder);
        const res = await listAll(listRef);
        this.photos = await Promise.all(
          res.items.map(async (item) => {
            const downloadURL = await getDownloadURL(item);
            return { url: downloadURL };
          })
        );
      } catch (error) {
        console.error("Error fetching photos: ", error);
      }
```

Slika 26. Prikaz koda za autentifikaciju i upravljanje datotekama

3. **Zaključak**

Ovaj rad opisuje proces izrade web aplikacije za kozmetičke salone koristeći modernu tehnologiju, poput HTML-a, CSS-a, JavaScript-a, Vue.js okvira i Google Firebase platforme. Razvijena aplikacija omogućuje administraciju sadržaja i upravljanje uslugama na jednostavan i učinkovit način, prilagođeno potrebama korisnika u industriji ljepote. Kroz implementaciju različitih funkcionalnosti, poput upravljanja fotografijama, kreiranja usluga i korisničke autentifikacije, ova aplikacija značajno unapređuje prezentaciju salona i poboljšava korisničko iskustvo.

Izrada ove aplikacije pokazala je kako upotreba odgovarajućih alata i tehnologija može rezultirati kvalitetnim rješenjem koje je lako prilagoditi specifičnim zahtjevima korisnika. Osim toga, ovaj projekt naglašava važnost integracije modernih tehnologija u svakodnevno poslovanje, čime se olakšava komunikacija s klijentima i povećava konkurentnost na tržištu. Daljnji razvoj aplikacije mogao bi uključivati dodatne funkcionalnosti, poput integracije s društvenim mrežama ili sustava za online rezervacije, čime bi se dodatno proširila korisnička baza i poboljšala efikasnost poslovanja salona.

Literatura[1] „Firebase Documentation“, Firebase Official Website. https://firebase.google.com/ (pristupljeno: 12.06.2024.)

[2] „Što je Node.js? Sve što trebate znati“, DIR.hr. https://dir.hr/sto-je-nodejs-sve-sto-trebate-znati/ (pristupljeno: 12.06.2024.)

[3] „History of JavaScript“, W3Schools. https://www.w3schools.com/js/js_history.asp (pristupljeno: 12.06.2024.)

[4] „How to Build a Single Page Application (SPA) with Vue.js“, ADCI Solutions. https://www.adcisolutions.com/knowledge/how-build-single-page-application-spa-vuejs (pristupljeno: 12.06.2024.)

[5] „Introduction to Node.js“, Node.js Official Documentation. https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (pristupljeno: 12.06.2024.)

[6] D. Bateman, „What is Firebase? The Complete Story (Abridged)“, Medium. https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 (pristupljeno: 12.06.2024.)

[7] „Vue CLI Configuration Reference“, Vue.js Official Documentation. https://cli.vuejs.org/config/ (pristupljeno: 12.06.2024.)

[8] „Vue.js: The Progressive JavaScript Framework“, Vue.js Official Documentation. https://vuejs.org/v2/guide/ (pristupljeno: 12.06.2024.)

[9] „CSS Flexbox and Grid - Layout Techniques for the Modern Web“, CSS-Tricks. https://css-tricks.com/snippets/css/a-guide-to-flexbox/ (pristupljeno: 12.06.2024.)

[10] S. Lee, „Designing User Interfaces with Figma: A Comprehensive Guide“, UX Design Journal. https://uxdesignjournal.com/figma-comprehensive-guide (pristupljeno: 12.06.2024.)

[11] D. Raggett, „A history of HTML“. https://www.w3.org/People/Raggett/book4/ch02.html (pristupljeno: 20.06.2024.)

[12] R. Thompson, „Understanding Firebase: Real-time Database and Beyond“, Coding Academy. https://codingacademy.com/firebase-guide (pristupljeno: 20.06.2024.)

[13] M. Jordan, „Best Practices in Modern Web Development: From HTML5 to Vue.js“, DevHub. https://devhub.com/articles/best-practices-modern-web-development (pristupljeno: 24.6.2024.)

[14] „JavaScript Overview - MDN Web Docs Glossary: Definitions of Web-related terms“, MDN Web Docs. https://developer.mozilla.org/en-US/docs/Glossary/JavaScript (pristupljeno: 24.06.2024.)

[15] A. Patel, „From Frontend to Full Stack: The Evolution of Web Development Technologies“, TechTonic. https://techtonic.com/frontend-fullstack-evolution (pristupljeno: 18.07.2024.)

[16] „Web Performance Optimization with Vue.js“, Performance Matters. https://performancematters.com/vuejs-optimization (pristupljeno: 22.07.2024.)

[17] J. Walker, „Developing Secure Web Applications: A Firebase Approach“, Security Today. https://securitytoday.com/firebase-secure-applications (pristupljeno: 28.07.2024.)

[18] „EmailJS Documentation“, EmailJS. https://www.emailjs.com/docs/ (pristupljeno: 04.08.2024.)

Popis slika