Internet aplikacija za pretraživanje i oglašavanje smještaja

Bukulin, Borna

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:961255

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-20**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni preddiplomski studij

INTERNET APLIKACIJA ZA PRETRAŽIVANJE I OGLAŠAVANJE SMJEŠTAJA

Završni rad

Borna Bukulin

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 14.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Borna Bukulin
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4324, 22.07.2019.
OIB Pristupnika:	47285295314
Mentor:	izv. prof. dr. sc. Josip Balen
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Internet aplikacija za pretraživanje i oglašavanje smještaja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	U teorijskom dijelu rada potrebno je proučiti i opisati zahtjeve za razvoj aplikacija za pretraživanje i oglašavanje smještaja i tehnologije za izradu web aplikacija. U praktičnom dijelu rada potrebno je izraditi web aplikaciju sa sučeljem u kojem će korisnik moći pretražiti oglašene smještaje i oglase za potragom sustanara uz mogućnosti sortiranja i filtriranja oglasa, oglasiti vlastiti smještaj uz mogućnost dodavanja,
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	14.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Borna Bukulin
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. studenta, godina upisa:	R 4324, 22.07.2019.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Internet aplikacija za pretraživanje i oglašavanje smještaja**

izrađen pod vodstvom mentora izv. prof. dr. sc. Josip Balen

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 1

2. METODE I TEHNIKE RADA 2

2.1. HTML, CSS, JavaScript 3

2.2. Uvod u MERN arhitekturu 3

2.3. MongoDB baza podataka 4

2.4. Node.js razvojna okolina 5

2.5. Express razvojni okvir 6

2.6. React JavaScript biblioteka 8

2.6.1. React komponente u virtualnom DOM-u 9

2.7. Tailwind CSS razvojni okvir 11

3. REALIZACIJA INTERNET APLIKACIJE 12

3.1. Podatkovni model internet aplikacije 12

3.2. Postavke razvojnog okruženja 15

3.3. MongoDB Atlas servis 17

3.4. Razvoj korisničkog sučelja 17

3.4.1. Upravljanje globalnim stanjem aplikacije 18

3.4.2. Upravljanje rutama aplikacije 22

3.5. Razvoj pozadinske programske podrške 24

3.5.1. Postavke i konfiguracija poslužitelja 25

3.5.2. Autentifikacija korisnika 26

3.5.3. Realizacija podatkovnog modela aplikacije 27

3.6. Konačna internet aplikacija 29

4. ZAKLJUČAK 34

LITERATURA 35

SAŽETAK 36

ABSTRACT 37

ŽIVOTOPIS 38

1. UVOD

U suvremenom svijetu, sveprisutna migracija mladih prema urbanim sredinama radi obrazovanja i zapošljavanja donosi sa sobom niz kompleksnih izazova. U većini slučajeva, prilikom pronalaska smještaja u novoj sredini, mladi su suočeni s izazovima visoke najamnine, sigurnosnih pologa i pregovora s nesusretljivim iznajmljivačima. Međutim, pronalazak idealnog smještaja predstavlja dugotrajan proces, dok odabir sustanara koji ima različite navike i interese vrlo brzo može postati izvor stresa i frustracije.

Pronalazak povoljnog smještaja, pronalazak osobe kojoj se može vjerovati kao budućem sustanaru te velika količina nepotpunih oglasa na nepreglednim Facebook grupama predstavljaju samo neke od glavnih izazova s kojima se ciljana skupina korisnika suočava kada je riječ o pronalasku smještaja u novoj sredini. Zajednički smještaj nameće se kao jedino prihvatljivo rješenje za studente i mlade profesionalce koji nailaze na isti problem. Korisnici moraju moći jednostavno, brzo i sigurno pronaći idealni smještaj ili odgovarajućeg sustanara, na željenoj lokaciji i po pristupačnoj mjesečnoj najamnini.

Glavno poglavlje rada uspoređuje postojeća digitalna rješenja na tržištu, kao i razloge odabira MERN arhitekture te prednosti svake od tehnologija i alata korištenih pri izradi internet aplikacije. Proučena je važnost modeliranja podatkovnog modela u NoSQL bazama podataka. Prikazana je implementacija upravljanja globalnim stanjem unutar React aplikacija korištenjem React Context API. Na primjeru je objašnjena važnost implementacije zaštićenih ruta za kontrolu pristupa internet aplikaciji. Postavljene su temelje značajke autentifikacije korisnika unutar internet aplikacije i realizacija podatkovnog modela. U konačnici, snimke zaslona internet aplikacije prikazuju rezultat implementacijskog dijela završnog rada.

1.1. Zadatak završnog rada

Zadatak ovog rada jest razvoj internet aplikacije s fokusom na pretraživanje i oglašavanje smještaja. Aplikacija pruža korisnicima jednostavno korisničko iskustvo (eng. *user experience*) u složenom procesu pronalaska idealnog stambenog rješenja. Za izradu ovog rada korištena je MERN arhitektura koja uključuje MongoDB platformu za pohranu podataka o korisnicima i stambenim objektima, Node.js i Express razvojni okvir za izradu pozadinskog programskog rješenja te React JavaScript biblioteku za izradu jednostavnog i intuitivnog sučelja. Stilizaciji korisničkog sučelja pristupljeno je s pažnjom prema detaljima, koristeći Tailwind CSS biblioteku kako bi se osiguralo koherentno i privlačno vizualno iskustvo.

2. METODE I TEHNIKE RADA

U razvoju modernih internet aplikacija, MERN arhitektura ističe se kao snažan temelj za izradu fleksibilnih i učinkovitih rješenja. Akronim MERN predstavlja kombinaciju MongoDB, Express, React i Node.js tehnologija koja promiče razvoj cjelovite (eng. *full-stack*) internet aplikacije koristeći isključivo JavaScript kôd na klijentskoj i poslužiteljskoj strani. Ovakav pristup omogućuje inženjerima samostalni razvoj cjelovitih aplikacija bez potrebe za uvođenjem novih programskih jezika. Napretkom tehnologije, ali i većom potrebom za rješavanjem sve složenijih korisničkih zahtjeva, internet aplikacije s jednom stranicom (eng. *SPA - single page application*) postale su standard. U kontekstu ovakvih aplikacija, korisničko sučelje ostaje aktivno, ažurirajući samo one dijelove koji zahtijevaju promjenu. Za svaki korisnički upit šalje se poziv poslužitelju koji kao odgovor vraća tražene podatke ili dio podataka, nakon čega se isključivo ciljani dio aplikacije ažurira novim sadržajem. Ovakav pristup znatno poboljšava cijelo korisničko iskustvo s naglaskom na optimizaciju izvođenja [1]. U narednim sekcijama ovog poglavlja detaljnije su opisane komponente MERN arhitekture, primjeri koda i njihova međusobna povezanost s ciljem izrade cjelovite internet aplikacije.

U pogledu već postojećih internet aplikacija na tržištu, koje djelomično rješavaju probleme navedene u uvodu završnog rada, ističu se SpareRoom, Student.com i Roomi platforme. Sve tri aplikacije nude korisnicima mogućnost jednostavnog i učinkovitog pretraživanja različitih opcija smještaja prilagođenih studentskim potrebama. Sličnosti među njima uključuju detaljne profile smještaja i korisnika te mogućnost ostavljanja recenzija i ocjena. Ipak, razlike između ovih platformi su značajne. Student.com se ističe svojom globalnom prisutnošću i prilagođenim uslugama međunarodnim studentima, čime olakšava pronalazak smještaja u inozemstvu. SpareRoom nudi širok raspon smještajnih opcija, uključujući dijeljenje stanova, što pruža fleksibilnost u izboru smještaja. S druge strane, Roomi je usredotočen na povezivanje studenata s istim interesima i životnim stilovima kako bi zajedno iznajmljivali smještaj, nudeći alate za organiziranje financija i komunikaciju među cimerima. Završni rad opisuje realizaciju internet aplikacije koja se fokusira na lokalno tržište. Kroz intuitivno korisničko sučelje korisnici imaju mogućnost pretraživanja, filtriranja i sortiranja dostupnih oglasa. Uvid u detalje poput lokacije, kvadrature smještaja, raspoloživih prostorija, broja trenutnih stanara kao i fotografije smještaja predstavljaju samo dio informacija dostupnih korisnicima. Nakon pronalaska idealnog smještaja ili odgovarajućeg sustanara, korisnici mogu izravno stupiti u kontakt s oglašivačem. Aplikacija također omogućuje korisnicima postavljanje vlastitog oglasa uz mogućnost naknadne izmjene i brisanja istoga.

2.1. HTML, CSS, JavaScript

Neizostavni dio svake internet stranice predstavlja HTML (eng. *Hypertext Markup Language*), standardizirani semantički jezik za izradu internet stranica. Primarna upotreba je izrada hipertekstualnih dokumenata. HTML elementi govore internet pregledniku kako prikazati sadržaj koristeći osnovne elemente pod nazivom znakovi (eng. *tags*). U izradi završnog rada korišten je HTML5, novija verzija HTML standarda. U odnosu na prethodnu verziju, HTML5 podržava audio i video kontrole, omogućuje pokretanje JavaScript-a u pozadini, pruža podršku za SVG sadržaj kao i veću prilagodljivost mobilnim uređajima, navodi službena HTML dokumentacija [2].

CSS (eng. *Cascading Style Sheets*) predstavlja stilski jezik za prikaz i prezentaciju izgleda same internet stranice. Uvođenjem CSS-a u već postojeću strukturu zasnovanu na HTML-u, nastoji se odvojiti prezentacija od sadržaja, što u konačnici rezultira boljom pristupačnosti i konzistentnim dizajnom sučelja. Kada je riječ o dizajnu obraća se pozornost na tipografiju, paletu boja, razmještaj elemenata, oblik elemenata i mnoge druge aspekte koje je moguće definirati pomoću CSS-a. Ipak, CSS može biti vremenski zahtjevan za pisanje te kompliciran za otklanjanje pogrešaka. U svrhu rješavanja ovog problema, danas se uglavnom umjesto tzv. *vanilla* CSS-a koriste pouzdane razvojne biblioteke, navodi Noel Rappin [3]. Razlike između popularnih razvojnih biblioteka za stiliziranje korisničkog sučelja kao i glavne značajke Tailwind CSS biblioteke, korištene pri izradi ovog završnog rada, detaljnije su opisane u potpoglavlju 2.7.

Istraživanjem provedenim 2023. godine, JavaScript predstavlja najzastupljeniji skriptni programski jezik u industriji [4]. JavaScript omogućuje viši stupanj interaktivnost između korisnika i internet aplikacije. Mnoge biblioteke namijenjene razvoju internet aplikacija zasnivaju se upravo na JavaScript programskom jeziku, poput React JavaScript biblioteke korištene u izradi rada. Više o samoj React biblioteci opisano je u potpoglavlju 2.6.

2.2. Uvod u MERN arhitekturu

Kao što je navedeno u uvodu poglavlja, MERN arhitektura predstavlja jednu od popularnijih arhitektura kada je riječ o razvoju cjelovitih dinamičkih internet aplikacija. Ova popularnost proizlazi iz prirode otvorenog koda arhitekture i jednostavnosti integracije između klijenta i poslužitelja. Sve komponente MERN arhitekture tehnički se temelje na JavaScriptu, pri čemu svaka od komponenti igra ključnu ulogu u razvoju cjelovitih internetskih aplikacija.

MERN arhitekturu možemo detaljnije razložiti u tri glavna sloja:

1. baza podataka – prvi sloj arhitekture odgovoran je za pohranu podataka u MongoDB bazu
2. poslužiteljski sloj – drugi sloj djeluje kao spona između klijentskog sloja i baze podataka, čine ga dvije ključne komponente: Express razvojni okvir i Node.js razvojna okolina
3. klijentski sloj – treći i posljednji sloj predstavlja korisničko sučelje internetske aplikacije, izrađeno korištenjem React biblioteke i stilizirano Tailwind CSS razvojnim okvirom.

U kombinaciji, navedeni slojevi MERN arhitekture omogućuju izgradnju skalabilnih, učinkovitih i dinamičkih internetskih aplikacija koje jednostavno povezuju korisničko sučelje, poslužitelja i bazu podataka.

2.3. MongoDB baza podataka

MongoDB baza podataka predstavlja dokumentno - orijentiranu bazu podataka opće namjene, suprotnu tradicionalnim relacijskim bazama podataka [5]. Ovakav pristup pruža veći stupanj fleksibilnosti kao i mogućnost jednostavnijeg skaliranja sa značajkama poput sortiranja, agregacije i geoprostornih indeksa. Ovo potpoglavlje detaljnije opisuje razlike između relacijskih i nerelacijskih baza podataka, kao i ključne značajke MongoDB baze podataka.

Relacijske baze podataka i sustavi koji omogućuju upravljanje istima, SURBP (eng. *RDBMS – Relational Database Management System*), koriste SQL (eng. *Structured Query Language*). Struktura entiteta u takvim bazama podataka vrlo je jasno i strogo definirana. Suprotno tome, NoSQL baze podataka ne prate tradicionalni relacijski model organizacije podataka, stoga se smatraju nerelacijskim bazama podataka. Zamjenjujući koncept tablice i reda s konceptom dokumenta u slučaju MongoDB baze podataka, postiže se mogućnost implementacije složenije hijerarhijske strukture s pojedinačnim zapisom. Nerelacijske baze podataka pružaju mogućnost strukturiranih, polustrukturiranih ili nestrukturiranih podataka. Tri su ključne značajke nerelacijskih baza podataka, kako navode S. Bradshaw, E. Brazil i K. Chodorow [5]:

- shema – kod nerelacijskih baza podataka ne postoji unaprijed definirana shema, što znači da niti jedan dokument nema fiksne vrijednosti ili ključeve
- računalna podrška – kada je riječ o nerelacijskim bazama podataka, pruža se mogućnost korištenja većeg broja cjenovno prihvatljivijih poslužitelja za obradu većeg broja podataka
- visoka distributivnost – podaci se mogu pohranjivati i obrađivati na više poslužitelja.

Kao što je navedeno ranije, MongoDB izdvaja se po ključnim značajkama koje čine ovu bazu podataka vrlo fleksibilnom i učinkovitom za pohranu te upravljanje podacima. Glavni strukturni element MongoDB baze podataka predstavlja dokument. Struktura MongoDB dokumenta vrlo je slična strukturi objekta u programskom jeziku JavaScript, zbog toga što je strukturno u JSON (eng. *JavaScript Object Notation*) formatu. Na slici *2.1*. prikazan je primjer dokumenta u MongoDB bazi podataka.

Slika 2.1. Primjer dokumenta u MongoDB bazi podataka.

MongoDB pohranjuje dokumente u kolekcije, što je analogno tablicama u tradicionalnim relacijskim bazama podataka. U ovom slučaju možemo primijetiti fleksibilnost MongoDB baze kada je riječ o upravljanju kolekcijama - ako određena kolekcija ne postoji, MongoDB stvara novu kolekciju prilikom prve pohrane podataka. Brza pohrana i pretraga podataka postižu se učinkovitom arhitekturom i indeksiranjem, dok napredan skup upita omogućuje precizno filtriranje, sortiranje i agregacije podataka. U konačnici, gotovo svaki aspekt MongoDB baze dizajniran je s fokusom na održavanje visoke razine performansi. Iako je MongoDB snažan alat, te uključuje mnoge značajke prisutne u relacijskim bazama podataka, važno je napomenuti da nije dizajniran za sve scenarije koje relacijske baze podržavaju [5].

2.4. Node.js razvojna okolina

U nizu programskih jezika koji se primjenjuju na strani poslužitelja, među najpopularnijima su Java, PHP, GO i Ruby. Tradicionalno, JavaScript se koristio isključivo na klijentskoj strani za omogućavanje interaktivnosti između korisnika i korisničkog sučelja. JavaScript programski jezik svoj je put do poslužitelja pronašao 2009. godine kada je Node.js (u daljnjem tekstu Node) preuzeo snažni Google V8 *engine* koji pokreće najpopularniji internet preglednik, Google Chrome. Na taj način omogućeno je izvođenje JavaScript kôda na strani poslužitelja.

U nastavku slijedi popis prednosti koje donosi odabir Node razvojne okoline za komunikaciju s poslužiteljem, prema Evanu M. Hahnu [6]:

- Node koristi Google V8 *engine* koji može izvršiti tisuće zahtjeva u sekundi
- zagovara se asinkroni stil pisanja koda s ciljem izbjegavanja problema s višenitnom obradom
- pruža pristup mnogim JavaScript bibliotekama otvorenog koda (eng. *open-source*)
- omogućuje dijeljenje koda između preglednika i poslužitelja s obzirom na to da je riječ o istom programskom jeziku.

Upravljač paketa za Node okolinu naziva se npm (eng. *node package manager*). Kao što samo ime upućuje, njegova svrha je upravljanje novim te nadogradnja postojećih Node paketa koji se koriste u internet aplikacijama, bilo na strani klijenta ili poslužitelja. Upravljač paketa olakšava upravljanje vanjskim JavaScript bibliotekama koje su otvorenog koda, besplatne i dostupne svima na korištenje. Takav pristup ubrzava razvoj internet aplikacija jer je moguće koristiti već gotova i provjerena rješenja za određene probleme. Značajan npm paket koji igra ključnu ulogu u izradi ovog rada je Express. Node API (eng. *Application Programming Interface*) može biti opsežan, kompleksan te u nekim slučajevima ograničen u pogledu značajki. Express predstavlja razvojni okvir koji djeluje kao dodatan sloj na Node pozadinsku programsku podršku, uvelike olakšavajući proces pisanja koda na strani poslužitelja.

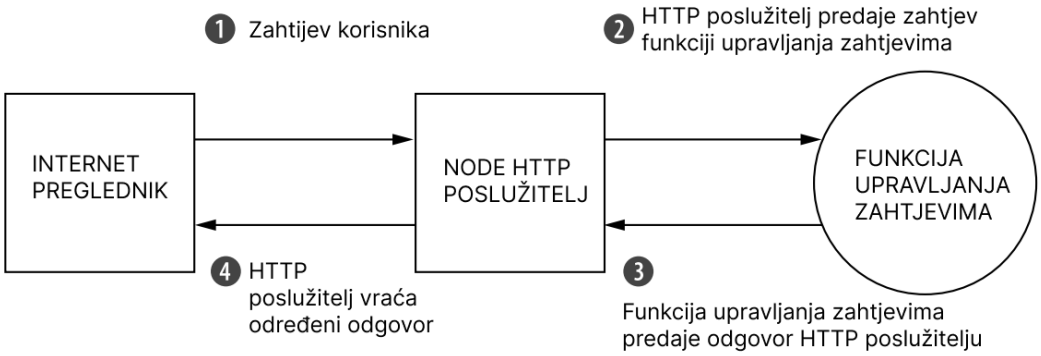
2.5. Express razvojni okvir

Express predstavlja minimalan i fleksibilan razvojni okvir ispod kojeg se nalazi Node razvojna okolina. Pruža znatno jednostavnije pozadinsko programsko sučelje obogaćeno mnogim značajkama koje nisu dostupne u izoliranoj Node okolini [6]. Iako je moguće implementirati cjelokupnu funkcionalnost na strani poslužitelja korištenjem tzv. *vanilla* JavaScripta na Node okolini, takav pristup može postati relativno kompliciran. Razlog tome leži u činjenici da osnovni moduli Node okoline nisu nužno optimizirani za ubrzani razvoj internet aplikacija. Express razvojni okvir svodi se na četiri temeljne značajke [6]:

- međuprogram (eng. *middleware*) – predstavlja pojam koji nije specifičan za Express razvojni okvir. Ideja međuprograma je da se umjesto jedne monolitne funkcije upravitelja zahtjevima poziva nekoliko funkcija upravitelja zahtjevima od kojih se svaka bavi specijaliziranim dijelom posla

- napredno usmjeravanje (eng. *routing*) – poput međuprograma, usmjeravanje razbija jednu monolitnu funkciju upravitelja zahtjevima na manje dijelove. Za razliku od međuprograma, ovi upravitelji zahtjevima izvršavaju se uvjetno, ovisno o URL-u i HTTP metodi koju klijent šalje. Usmjeravanje omogućuje razdvajanje ponašanja aplikacije ovisno o ruti
- podaplikacije (eng. *subapplications*) – kada je riječ o skaliranju internet aplikacija napisanih u Express razvojnom okviru, podaplikacije predstavljaju vrlo korisnu značajku. Pisanje Express podaplikacija je gotovo isto kao i pisanje aplikacija normalne veličine, ali omogućuje dodatno dijeljenje aplikacije na manje dijelove
- pogodnosti (eng. *conveniences*) – predstavljaju niz apstraktnih i korisnih značajki koje znatno smanjuju složenost Node razvojne okoline. Express pogodnosti konceptualno ne mijenjaju način na koji je aplikacija organizirana.

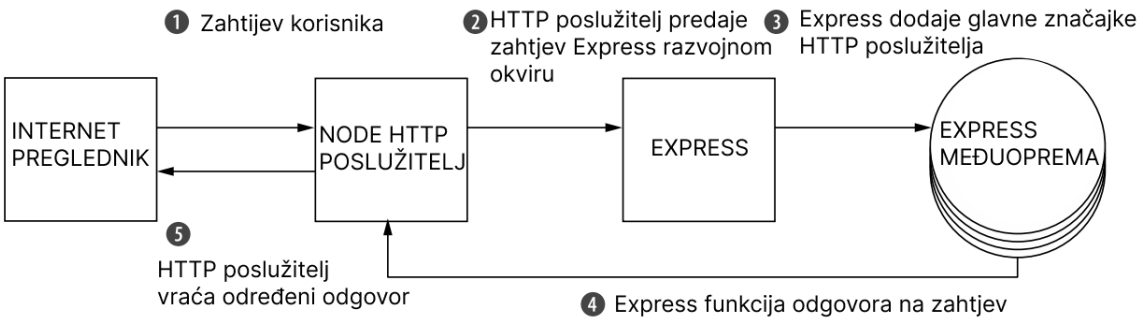
Prilikom izrade pozadinske programske podrške u Nodeu implementira se jedna JavaScript funkcija za cijelu aplikaciju, tzv. funkcija upravljanja zahtjevima (eng. *request handler*). Njezin zadatak je slušati zahtjeve internet preglednika koji komunicira s pozadinskim programskim sučeljem aplikacije. U trenutku kada internet preglednik pošalje zahtjev, jedinstvena funkcija pregledava zahtjev i vraća određeni traženi odgovor s poslužitelja. Primjerice, jedini zadatak internet aplikacije je vratiti korisniku traženi podatak s poslužitelja. Traženi podatak se nalazi na početnoj stranici internet aplikacije. Značajke koje je potrebno implementirati pozadinskom programskom podrškom jesu vraćanje traženog podatka na početnoj stranici internet aplikacije i vraćanje HTTP 404 *Not Found* greške ako korisnik preda zahtjev za bilo koju drugu krajnju točku. Tijek zahtjeva pozadinske programske podrške napisan isključivo u Node razvojnoj okolini bez Express razvojnog okvira prikazan je na slici *2.2.*



Slika 2.2. Tijek zahtjeva Node pozadinske programske podrške.
Moguće je utjecati na kružne oblike, kvadratni oblici su izvan domene utjecaja [6].

Node HTTP poslužitelj upravlja vezom između klijenta i JavaScript funkcije stoga nije potrebno dodatno raditi s mrežnim protokolima. Problem kod ovog pristupa javlja se ukoliko je riječ o većim internet aplikacijama koje imaju kompleksnije pozadinsko sučelje. Za usporedbu, slika *2.3.* prikazuje tijek zahtjeva Express pozadinske programske podrške.

Umjesto jedne velike funkcije koja upravlja zahtjevima, Express nudi mogućnost pisanja većeg broja manjih funkcija, koje također mogu biti funkcije vanjske biblioteke. Neke funkcije se izvode na svaki zahtjev, poput funkcije koja bilježi sve zahtjeve, dok se druge funkcije izvode nekolicinu puta, na primjer funkcija koja upravlja HTTP 404 *Not Found* greškom.



Slika 2.3. Tijek zahtjeva Express pozadinske programske podrške [6].

2.6. React JavaScript biblioteka

Nastao 2013. godine pod vodstvom Facebooka, React danas predstavlja najpopularniju biblioteku za razvoj korisničkih sučelja, prema rezultatima Stack Overflow ankete iz 2023. godine [7]. Osim što je biblioteka otvorenog koda, glavne prednosti Reacta leže u performansama i fleksibilnosti koja omogućuje stvaranje elemenata za višekratnu upotrebu. Takvim pristupom osigurano je pisanje koda koji je jednostavan za održavanje i proširivanje. Osnovne atomske jedinice React biblioteke predstavljaju komponente (eng. *components*). Komponente su jednostavni višekratni elementi koji unutar sebe sadrže cijelu svoju funkcionalnost koja ima isključivo jednu odgovornost.

Prilikom rada s React bibliotekom gotovo je neizbježno koristiti JSX (eng. *JavaScript XML*) proširenje JavaScript programskog jezika. Sintaksa JSX-a podsjeća na sintaksu korištenu u HTML-u. Zbog ograničenosti internet preglednika u interpretaciji isključivo JavaScript programskog jezika, JSX se prilikom prevođenja aplikacije pretvara u odgovarajući JavaScript kod. Za automatsko prevođenje zadužen je JavaScript prevoditelj Babel.

Prema navodima službene React dokumentacije [8], logika prikazivanja komponenti čvrsto je vezana uz ostale logike korisničkog sučelja, poput upravljanja različitim događajima (eng. *event handlers*), promjene stanja tijekom vremena te pripreme podataka za prikaz. React ne zahtjeva korištenje JSX-a, ali se pokazalo kako ga većina inženjera smatra korisnim kao vizualnu pomoć pri radu s korisničkim sučeljem unutar JavaScript koda.

2.6.1. React komponente u virtualnom DOM-u

Konceptualno, komponente su poput JavaScript funkcija. Imaju mogućnost prihvaćanja proizvoljnih ulaznih argumenata (eng. *properties, props*) i vraćaju React elemente. Za razliku od funkcijske komponente, komponenta kao klasa u React biblioteci zahtijeva proširivanje istoga, kao i upotrebu drugačije sintakse u pisanju koda [9]. Izlaskom verzije 16.8. u kojoj su predstavljene tzv. *hook* funkcije, komponente kao klase gube na svojoj popularnosti.

Hook funkcije predstavljaju specijalizirane metode pomoću kojih je moguće koristiti stanja (eng. *states*) i metode životnog ciklusa (eng. *lifecycle methods*) unutar funkcijskih komponenti. Programski kod 2.1. prikazuje primjer funkcijske komponente korištene u radu. Riječ je o komponenti *SearchRow.jsx*, koja kao argumente prima *type, name, value, handleChange* i *labelText* podatke.

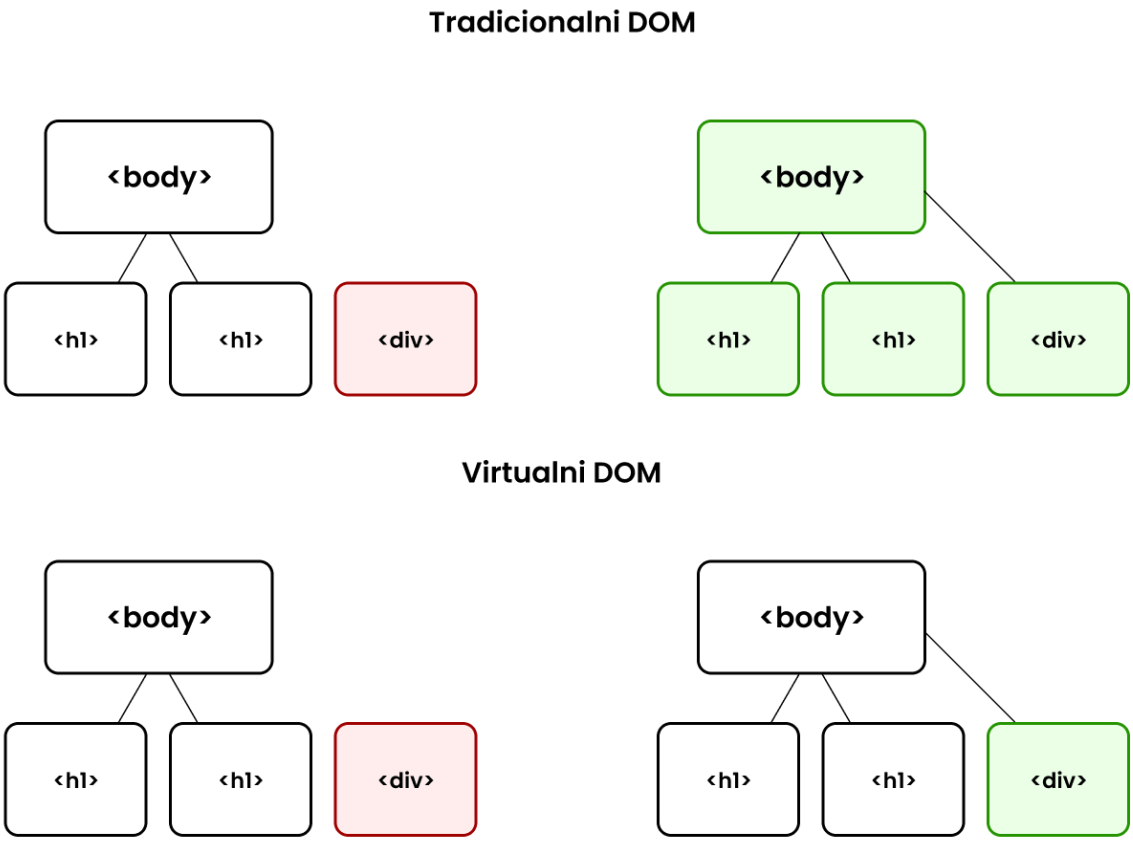
```
const SearchRow = ({ type, name, value, handleChange, labelText }) => {
return (
		<div>
			<label
				className="block text-sm font-medium text-neutral-700"
				htmlFor={name}>
				{labelText || name}
			</label>
			<input
				type={type}
				value={value}
				name={name}
				onChange={handleChange}

className="block w-full rounded-md border-2 border-neutral-200 bg-neutral-50 py-2 px-3 text-
neutral-800 placeholder-neutral-400 shadow-sm transition duration-500 hover:border-primary-400
hover:shadow-md focus:border-primary-500 focus:outline-none focus:ring-primary-500 sm:text-sm"
			/>
		</div>
	);
};

export default SearchRow;
```

Programski kod 2.1. Primjer funkcijske komponente korištene u radu.

Internet aplikacije zasnivaju se na sučelju za organizaciju svih prikazanih elemenata stranice, tzv. DOM (eng. *Document Object Model*). Dokumentni objektni model ima strukturu stabla koja pruža mogućnost brzog pretraživanja i upravljanja elementima. React biblioteka svojim virtualnim

DOM-om (eng. *virtual DOM*) kreira točnu kopiju odgovarajućeg stvarnog DOM-a. Također predstavlja stablo čvorova koje se sastoji od elemenata, njihovih atributa i drugih svojstava. Stablo čvorova ažurira se na temelju promjena koje se događaju u podatkovnom modelu, a uzrokovane su radnjama korisnika ili sustava. Ukoliko dođe do promjene u podatkovnom modelu, React uspoređuje novonastalo stanje virtualnog DOM-a sa stanjem prije promjene u podatkovnom modelu. Nakon uspješnog identificiranja ažuriranog elementa, dolazi do ažuriranja isključivo tog elementa u stvarnom DOM-u.

Na slici 2.4. jasno je prikazana razlika između tradicionalnog i virtualnog React DOM-a. Na lijevoj strani, prikazano je početno stanje DOM-a s *body* elementom koji sadrži dva *h1* elementa. Element koji se dodaje u DOM označen je crvenom bojom. Na desnoj strani slike prikazani su elementi koji se ponovno ažuriraju nakon što je dodan *div* element u DOM.



Slika 2.4. Usporedba tradicionalnog i virtualnog DOM-a.

2.7. Tailwind CSS razvojni okvir

Kao što je navedeno u prvom poglavlju, CSS omogućuje kontrolu prikaza informacija. Na samom početku interneta, CSS je uspješno ispunjavao sve zahtjeve inženjera. Eksponencijalnim porastom kompleksnosti internet aplikacija na korisničkom sučelju, uporaba izvornog, tzv. *vanilla* CSS-a bez pomoćnog razvojnog okvira može značajno usporiti brzinu razvoja internet aplikacije. Neki od najpopularnijih CSS razvojnih okvira i biblioteka su Bootstrap, Materialize i Tailwind CSS.

Tailwind CSS, prema službenoj dokumentaciji [10], predstavlja tzv. *utility – first* razvojni okvir. Bootstrap i slične biblioteke namijenjene izradi korisničkih sučelja pružaju CSS klase koje predstavljaju već gotove komponente. Primjerice, *button, card, nav* i mnoge druge. Takve klase imaju tendenciju unaprijed definirati niz CSS stilova. Suprotno tome, gotovo sve osnovne klase Tailwind razvojnog okvira su tanki sloj koji predstavlja jednu postavku CSS stila. Primjerice, umjesto klasa *font-size: 0.75rem;* i *line-height: 1rem;*, Tailwind pruža predefiniranu klasu *text-xs* koja postiže isti rezultat [11]. Tailwind kod je krajnje eksplicitan i omogućuje brzo razvijanje prototipa, brze iteracije i prilagodbe korisničkog sučelja.

Korištenje Tailwind razvojnog okvira zahtjeva pisanje manje CSS koda, rezultirajući vremenski efikasnijim razvojem korisničkog sučelja. Ne postoji potreba za pisanjem jedinstvenih imena varijabli kao što je slučaj u tzv. *vanilla* CSS-u, niti je potrebno upravljati globalnim stilom internet aplikacije. Vjerojatno najbitnija značajka Tailwind razvojnog okvira je velika fleksibilnost i prilagodljivost osnovnih klasa.

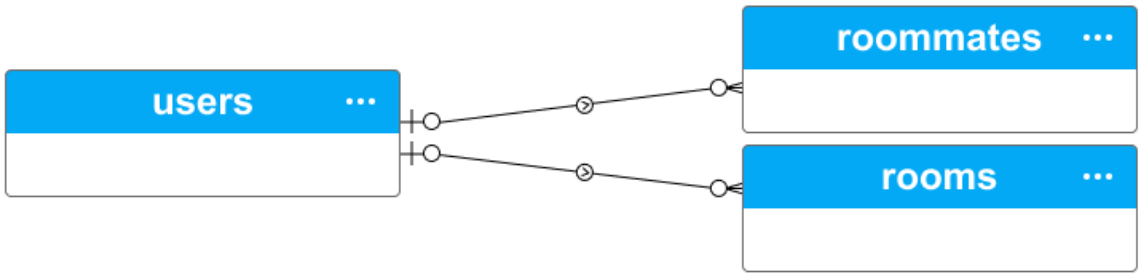
3. REALIZACIJA INTERNET APLIKACIJE

U nadolazećem poglavlju opisan je tijek izrade internet aplikacije za pretraživanje i oglašavanje smještaja i sustanara. Prezentirana je upotreba i konfiguracija korištenih alata, biblioteka, i razvojnih okolina. Cjelokupna internet aplikacija napisana je u Visual Studio Code integriranom razvojnom okruženju.

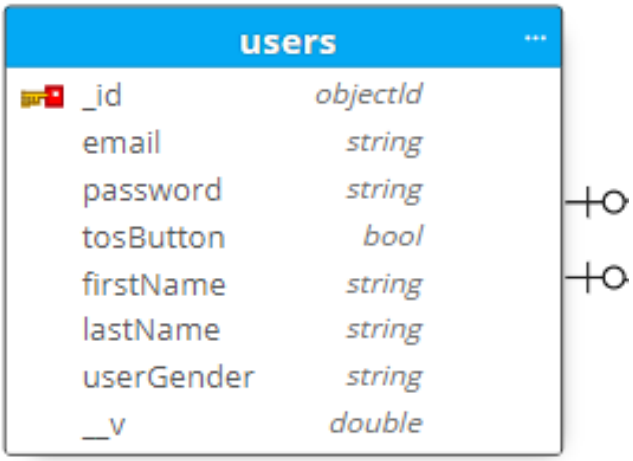
3.1. Podatkovni model internet aplikacije

Kao što je navedeno u poglavlju 2.3., NoSQL baze podataka nisu vođene strogo definiranom strukturom osnovnih elemenata. Glavna prednost MongoDB baze podataka leži u tome da njezina struktura nije čvrsto definirana, omogućujući tako učinkovit i fleksibilan razvoj aplikacije, za razliku od SQL baza podataka u kojima je potrebno prvo definirati ERA (eng. *entity-relationship-attribute*) model, stvoriti tablice i strogo slijediti model podataka kojim su iste definirane.

Međutim, velika fleksibilnost i sloboda kada je riječ o dizajnu podatkovnih modela NoSQL baza, ne znači nužno da podatkovni model ne mora pratiti dobre konvencije dizajna. Način na koji su podaci organizirani u dokumente i kolekcije izravno utječe na učinkovitost aplikacije u dohvaćanju i analiziranju podataka. Dakle, u svrhu održavanja strukture, smanjenja pogrešaka i omogućavanje čišćeg koda aplikacije, sheme su neophodne. U svrhu dizajniranja podatkovnog modela MongoDB baze podataka postoji niz pomoćnih alata. Za potrebe ovog rada korišten je alat pod nazivom Moon Modeler. Na slici *3.1.* prikazan je podatkovni model završnog rada, dok slika *3.2.* prikazuje detaljan model *users* dokumenta.



Slika 3.1.Visoka razina pregleda podatkovnog modela završnog rada.



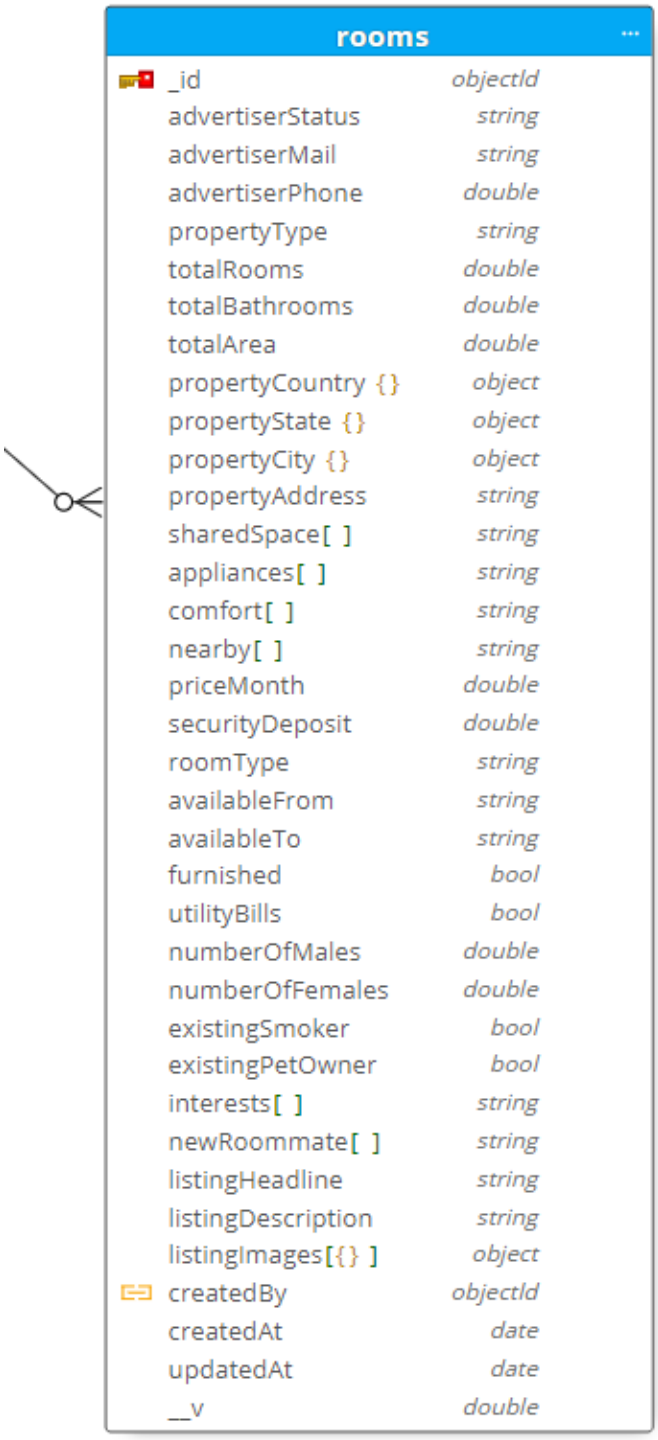
Slika 3.2. Detaljan model dokumenta korištenog za pohranu korisničkih podataka.

U nastavku slijedi opis modela *users* dokumenata:

- *email* – mail adresa korisnika
- *password* – kriptirana lozinka koristeći bcrypt npm paket
- *tosButton* – privola za uvjete korištenja usluge
- *firstName, lastName, userGender* – osnovni podaci o korisniku.

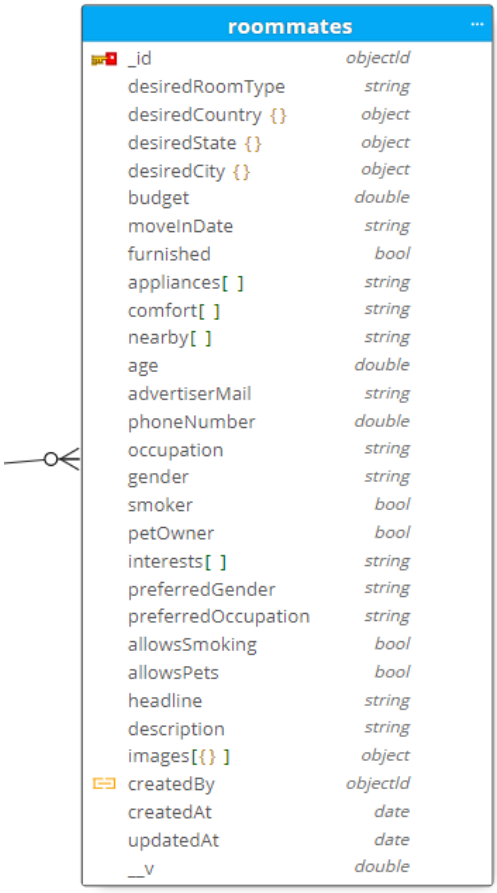
Analogno tome, slika *3.3*. prikazuje detaljan model *rooms* dokumenta, s objašnjenjem atributa koji potencijalno nisu razumljivi sami po sebi:

- *sharedSpace* – dijeljene prostorije u smještaju, predodređene vrijednosti uključuju Bathroom, Kitchen, Living room
- *appliances* – kućanski aparati i uređaji, predodređene vrijednosti uključuju Microwave, Stove, Television, Washer te Fridge
- *utilityBills* – označava jesu li režije uključene u cijenu smještaja
- *numberOfMales, numberOfFemales* – označava broj muških i ženskih sustanara koji trenutno žive u oglašenom smještaju.



Slika 3.3. Detaljan model dokumenta korištenog za pohranu podataka o smještaju.

Detaljan model *roommates* dokumenta prikazan je na slici 3.4. Tipovi podataka u atributima većinom uključuju *string*, *double*, *date*, *bool* i *object*. Ističe se *objectId* kao poseban tip identifikatora sadržan u BSON (eng. *Binary JSON*) specifikaciji - binarni format za serijalizaciju dokumenata u MongoDB bazi. Upotreba *objectId* tipa kao jedinstvenog identifikatora donosi prednost jer zauzima vrlo malo memorijskog prostora i automatski je poredan prilikom zapisa. Ovaj pristup osigurava redoslijed kod svih sljedećih zapisa [12].



Slika 3.4. Detaljan model dokumenta korištenog za pohranu podataka o sustanarima.

3.2. Postavke razvojnog okruženja

Visual Studio Code integrirano razvojno okruženje predstavlja najpopularniji izbor kada je riječ o razvoju internet aplikacija [13]. Otvorenoga je koda, besplatan te nudi podršku za mnoge programske jezike, uključujući TypeScript, Java, Powershell i PHP. Repozitorij aplikacije organiziran je u dva zasebna podrepozitorija, *client* i *server*. Naredbom *npm init* u svakom od podrepozitorija inicijalizira se Node projekt koji omogućuje instalaciju vanjskih npm paketa.

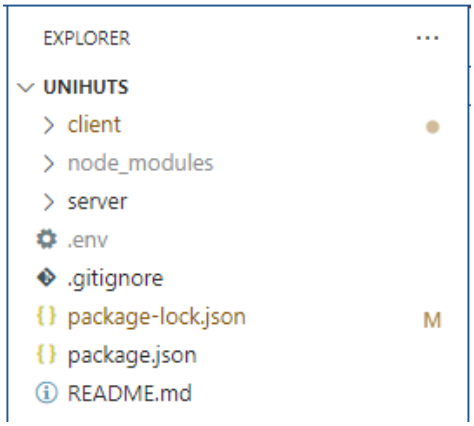
Slijedi popis ključnih vanjskih paketa instaliranih u client repozitoriju, uz kratak opis i svrhu njihove instalacije:

- *formik* – napredna biblioteka za upravljanje obrascima u React aplikacijama
- *yup* – validacijska biblioteka za provjeru ispravnosti podataka unesenih u obrasce
- *react-router-dom* – biblioteka za upravljanje rutama u React aplikacijama
- *axios* – biblioteka za izvršavanje HTTP zahtjeva, olakšava komunikaciju s poslužiteljem.

Prilikom inicijalizacije projekta fokusiranog isključivo na razvoj korisničkog sučelja, najčešće se koristi *create-react-app* CLI (eng. *command line interface*). Riječ je o skripti koja se izvršava u naredbenom retku. Upotrebom *create-react-app* skripte značajno se ubrzava postupak postavljanja novog React projekta na klijentskoj strani.

Slijedi kratak popis ključnih vanjskih paketa instaliranih u repozitoriju *server*, dok slika *3.5.* prikazuje konačnu strukturu glavnog direktorija:

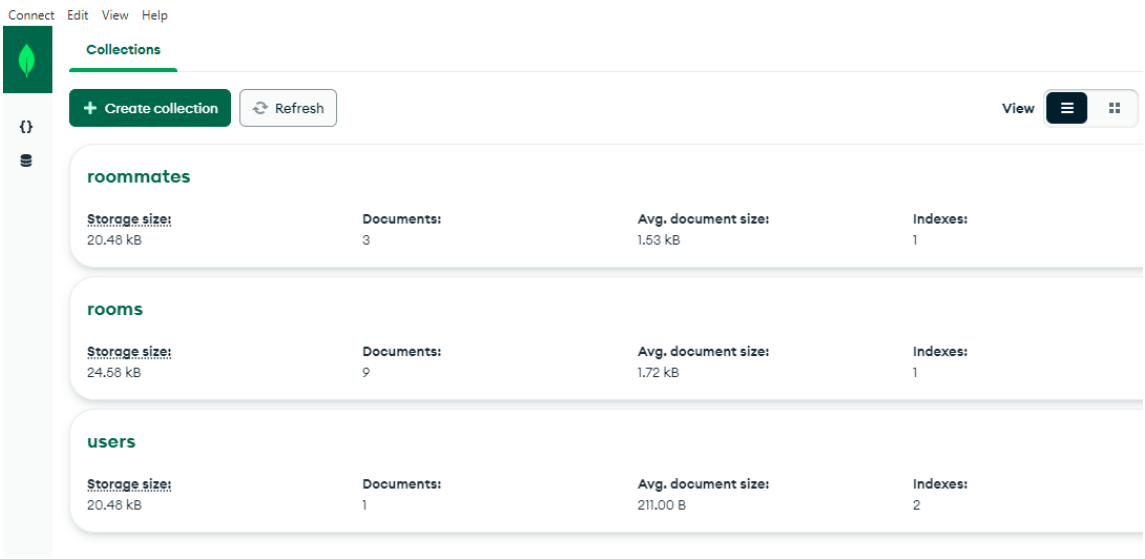
- *express* – glavna biblioteka korištena za izradu pozadinske programske podrške
- *jsonwebtoken* – biblioteka za generiranje i provjeru JSON Web Tokena (JWT) korištenog za autentifikaciju korisnika
- *mongoose* – ODM (eng. *Object Data Modeling*) biblioteka za jednostavno povezivanje i upravljanje MongoDB bazom podataka
- *bcryptjs* – biblioteka za sigurno kriptiranje i upravljanje lozinkama korisnika
- *body-parser* – međuprogram za obrađivanje dolaznih zahtjeva u Express aplikacijama
- *dotenv* – paket za učitavanje varijabli okruženja iz .env datoteke, konfiguracija okruženja.



Slika 3.5. Konačna struktura glavnog direktorija u aplikaciji.

3.3. MongoDB Atlas servis

MongoDB Atlas, u trenutku pisanja ovog rada, predstavlja vodeću uslugu tzv. *database-as-a-service*, baze podataka u oblaku (eng. *cloud*). Pruža neusporedivu mogućnost distribucije podataka i povezanost kroz AWS, Azure i Google Cloud servise [14]. Prilikom izrade rada korišten je MongoDB Atlas u kombinaciji s MongoDB Compass alatom. MongoDB Compass interaktivni je alat za optimizaciju i analizu MongoDB podataka [15]. Slika *3.6.* prikazuje korisničko sučelje MongoDB Compass alata na kojem je vidljivo ime baze podataka unihuts te pripadajuće kolekcije *roommates*, *rooms* i *users*.

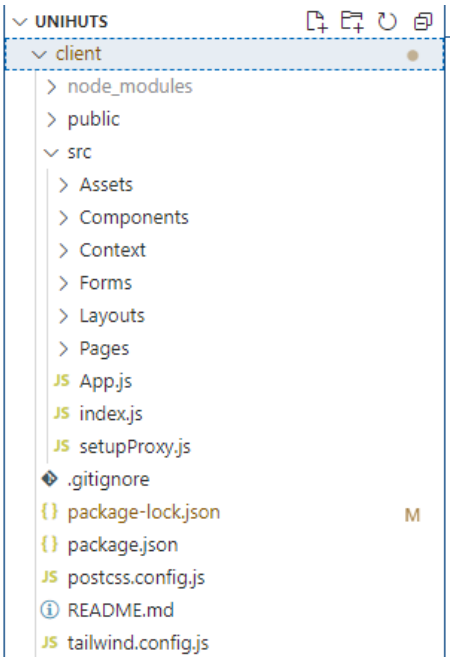


Slika 3.6. MongoDB Compass korisničko sučelje.

3.4. Razvoj korisničkog sučelja

Kao što je navedeno ranije, korisničko sučelje aplikacije izrađeno je React JavaScript bibliotekom, dok je projekt kreiran *create-react-app* skriptom. Slika *3.7.* prikazuje krajnju strukturu direktorija *client*. Slijedi kratak opis sadržaja koji se nalazi u poddirektoriju *client*:

- *node_modules* – sadrži datoteke svih instaliranih i korištenih vanjskih paketa
- *public* – sadrži statičke datoteke koje servira Express pozadinska programska podrška
- *src* – najvažnija mapa projekta iz razloga što sadrži datoteke izvornog koda. Uključuje podmape *Assets* za SVG ilustracije i ikone, *Components* s višekratnim React komponentama, *Context* koji je zadužen za upravljanje stanjem aplikacije te *Forms*, *Layouts* i *Pages* koji također sadrže iskoristive React komponente
- *index.js* i *App.js* – predstavljaju glavnu ulaznu točku internet aplikacije.



Slika 3.7. Konačna struktura client direktorija unutar aplikacije.

3.4.1. Upravljanje globalnim stanjem aplikacije

Globalno stanje aplikacije predstavlja vrlo važan aspekt u razvoju internet aplikacija s jednom stranicom. Važno je napomenuti da se podaci, u tipičnoj React aplikaciji, prosljeđuju odozgo prema dolje, točnije od roditelja prema djetetu putem tzv. rekvizita (eng. *props*). Globalno stanje na vrlo praktičan i efikasan način omogućuje dijeljenje podataka kroz stablo komponenti, eliminirajući potrebu za eksplicitnim prosljeđivanjem podataka kroz svaku razinu stabla [16]. Ovaj način upravljanja globalnim stanjem aplikacije React uvodi u verziji 16.3.0., predstavljajući Context API. Osim što predstavlja efikasnije rješenje, uporaba Context-a osigurava izbjegavanje tzv. *prop drilling* načina pisanja koda. *Prop drilling* predstavlja lošu praksu jer se isti podaci šalju na gotovo svim razinama zbog zahtjeva na konačnoj razini, u većini slučajeva rezultirajući viškom programskog koda.

Globalno stanje kreira se naredbom *React.createContext()*, nakon čega se unutar njega propisuju svi podaci koji će se prosljeđivati Context Providerom. Globalno stanje upravlja podacima o trenutnom upozorenju (obavijesti), korisniku, oglasu i paginaciji. Unutar globalnog stanja postavljaju se i inicijalne ili početne vrijednosti varijabli. Programski kod 3.1. prikazuje dio strukture globalnog stanja kreiranog pomoću React Context API-a.

```
const AppContext = React.createContext();

const AppProvider = ({ children }) => {
	const [state, dispatch] = useReducer(Reducer, initialState);

	// * AXIOS
	const authFetch = axios.create({
		baseURL: "/api/v1",
	});

	// * REQUEST
	authFetch.interceptors.request.use(
		(config) => {
			config.headers.common["Authorization"] = `Bearer ${state.token}`;
			return config;
		},
		(error) => {
			return Promise.reject(error);
		}
	);

	// * RESPONSE
	authFetch.interceptors.response.use(
		(response) => {
			return response;
		},
		(error) => {
			if (error.response.status === 401) {
				logoutUser();
			}
			return Promise.reject(error);
		}
	);
```

Programski kod 3.1. Primjer funkcijske komponente korištene u radu.

Programski kod iznad prikazuje *AppProvider* komponentu koja igra ključnu ulogu u postavljanju globalnog stanja i upravljanju Axios zahtjevima. Unutar nje, korišten je React *hook useReducer* za upravljanje stanjem aplikacije, pri čemu *state* varijabla predstavlja trenutno stanje, dok *dispatch* omogućuje izvršavanje akcija za promjenu tog stanja. Osim toga, stvorena je Axios instanca nazvana *authFetch* koja se koristi za slanje HTTP zahtjeva prema pozadinskoj programskoj podršci. Presretač zahtjeva za autorizaciju postavlja *Authorization* zaglavlje sa tokenom iz globalnog stanja, omogućujući tako autentifikaciju korisnika pri svakom zahtjevu aplikaciji. Kreirano globalno stanje potom je potrebno proslijediti određenim komponentama na nižim razinama, što se može ostvariti koristeći Context Provider unutar kojega se nalaze funkcije zadužene za rukovanje podacima, što je vidljivo u programskom kodu 3.2.

```
        return (
                <AppContext.Provider
                        value={{
                                ...state,
                                displayAlert,
                                clearAlert,
                                registerUser,
                                loginUser,
                                logoutUser,
                                updateUser,
                                clearValues,
                                listRoom,
                                getMyRooms,
                                setEditRoom,
                                editRoom,
                                deleteRoom,
                                listRoommate,
                                getMyRoommates,
                                setEditRoommate,
                                editRoommate,
                                deleteRoommate,
                                getAllRooms,
                                getAllRoommates,
                                handleChange,
                                clearFilters,
                                changePage,
                                getSingleRoom,
                                getSingleRoommate,
                        }}>
                        {children}
                </AppContext.Provider>
        );
};

const useAppContext = () => {
        return useContext(AppContext);
};
```

Programski kod 3.2. useAppContext hook koji vraća globalno stanje aplikacije.

Programski kod iznad prikazuje dijeljene vrijednosti u globalnom stanju aplikacije, što uključuje trenutno stanje *state* te niz funkcija za upravljanje aplikacijom, kao što je prikazivanje obavijesti upozorenja, registracija, prijava i odjava korisnika, ažuriranje korisničkih podataka itd. Sve vrijednosti postavljene su kao kontekst vrijednosti *App.Context.Provider* komponente i dostupne svim komponentama koje koriste *useAppContext* prilagođeni *hook*. Ovakav pristup olakšava centralizirano upravljanje globalnim stanjem i funkcionalnostima. Uvozom *useAppContext hook*, na vrlo jednostavan način je moguće dobiti tražene podatke iz globalnog stanja aplikacije. Programski kod 3.3. demonstrira implementaciju *ListingsContainer* komponente čija je funkcionalnost prikazati sve aktivne oglase smještaja, ukupan broj oglasa, trenutnu stranicu paginacije, ukupan broj stranica te ponuditi opcije filtriranja i sortiranja oglasa.

```
import { useEffect } from "react";
import { motion, AnimatePresence } from "framer-motion";
import { useAppContext } from "../../Context/AppContext";

import PublicRoomCard from "./PublicRoomCard";
import Pagination from "../../Layouts/Pagination";
import RoomSearch from "../../Layouts/RoomSearch";
import CardSkeleton from "../Loaders/CardSkeleton";
import EmptyStateSearch from "../../Layouts/EmptyStateSearch";

function ListingsContainer() {
	const {
		getAllRooms,
		listingsRooms,
		totalRoomListings,
		totalRoomPages,
		currentPageRoom,

		isLoading,
		searchRoomProperty,
		searchRoomCity,
		searchRoomHood,
		searchRoomType,
		searchRoomMain,
		sortRoom,
	} = useAppContext();
...
```

Programski kod 3.3. ListingsContainter komponenta.

Globalno stanje aplikacije implementirano pomoću Context API-ja omogućuje i postavljanje početnih vrijednosti unutar aplikacije. Programski kod 3.4. prikazuje implementaciju dijela varijabli i njihove pripadajuće početne vrijednosti.

```
const initialState = {
	// * App
	isLoading: false,
	showAlert: false,
	alertText: "",
	alertType: "",
	user: user ? JSON.parse(user) : null,
	token: token,

	// * Room listing
	isEditingRoom: false,
	editRoomId: "",
	room: null,
	rooms: [],
	totalRooms: 0,

	// * Roommate listing
	isEditingRoommate: false,
	editRoommateId: "",
	roommate: null,
	roommates: [],
	totalRoommates: 0,
```

Programski kod 3.4. Dio početnih vrijednosti aplikacije.

Slijedi kratko objašnjenje početnih stanja:

- *showAlert, alertText, alertType* – koriste se za upravljanje prikazom upozorenja,
- *user, token* – čuvaju informacije o prijavljenom korisniku
- *isEditingRoom, editRoomId, room, rooms, totalRooms* – koriste se za upravljanje korisničkim oglasima i omogućujući njihovo uređivanje
- *listingsRooms, listingsRoommates* – čuvaju informacije o smještajima i stanarima za javno pregledavanje dostupno svim korisnicima aplikacije.

3.4.2. Upravljanje rutama aplikacije

Iako React ima svoje rješenje za upravljanje komponentama, ono nije izravno usmjereno na navigaciju i rute, stoga je za potrebe upravljanja rutama pri izradi ovog završnog rada korišten jedan od najpopularnijih izbora – npm paket *react-router-dom*. React Router omogućuje kreiranje velikog broja ruta te dinamičke izmjene prikazanih komponenti ovisno o trenutnoj ruti u URL-u. Također, dolazi s nizom komponenti koje olakšavaju upravljanje rutama. Za ispravnu funkcionalnost React Router paketa, potrebno je uvesti *BrowserRouter (Router)* komponentu u najvišu razinu aplikacije. Komponenta *Router* pomaže u sinkronizaciji korisničkog sučelja s URL-om, stoga se u aplikaciji mora postaviti iznad svake komponente koja koristi React Router. Programski kod 3.5. prikazuje dio implementacije React Routera.

Glavna funkcija *App* predstavlja temeljnu komponentu koja se prikazuje na svakoj ruti unutar aplikacije. *App* komponenta je omotana *Router* komponentom koja omogućava upotrebu React Routera. Unutar *Routes* komponente definiraju se različite rute i njima pridružene komponente koje će se prikazivati kada se korisnik nađe na određenoj ruti. Svaka ruta ima svoj *path* atribut koji određuje URL na kojem će se prikazati pridružena komponenta. Unutar glavne rute definirane s *path="/"*, niz *Route* komponenti određuje podrute koje će se prikazivati unutar *SharedLayout* komponente. Uz to, definirane su i parametrizirane rute poput *room/:roomId* i *roommate/:roommateId*, gdje varijable *roomId* i *roommateId* predstavljaju dinamičke parametre u URL-u. Isječak programskog koda 3.6. prikazuje implementaciju logike iza učitavanja detalja određenog smještaja na korisničkom sučelju koristeći parametrizirane rute.

```
function App() {
	return (
	<>
	<Router>
		<Routes>
			<Route path="/" element={<SharedLayout />}>
				<Route path="" element={<HomePage />} />
				<Route path="rooms" element={<RoomsListings />} />
				<Route path="room/:roomId" element={<RoomDetails />} />

				<Route path="roommates" element={<RoommatesListings />} />
				<Route path="roommate/:roommateId" element={<RoommateDetails/>}/>
				<Route path="about" element={<AboutPage />} />
				<Route path="attributions" element={<AttributionsPage />} />
			</Route>

				<Route path="/login" element={<LoginPage />} />
				<Route path="/register" element={<RegisterPage />} />
				<Route path="*" element={<NotFoundPage />} />

			<Route
				path="/"
				element={
					<ProtectedRoute>
						<SharedLayout />
					</ProtectedRoute>}>
					<Route path="my-rooms" element={<MyRoomsPage />} />
					<Route path="my-roommates" element={<MyRoommatesPage/>} />
					<Route path="settings" element={<SettingsPage/>} />
			</Route>
		</Routes>
	</Router>
		</>
	);
}
```

Programski kod 3.5. Dio implementacije React Router paketa u App.js datoteci.

```
export default function RoomDetails() {
	const params = useParams();
	const [room, setRoom] = useState(null);
	const [message, setMessage] = useState("");
	useEffect(() => {
		const getRoom= async () => { const response = await
		axios.get(`/api/v1/public/room/${params.roomId}`);
			setRoom(response.data.room);
		};
		getRoom();
		// eslint-disable-next-line }, []);

const onChange = (e) => setMessage(e.target.value);
```

Programski kod 3.6. Dio implementacije prikaza smještaja na korisničkom sučelju.

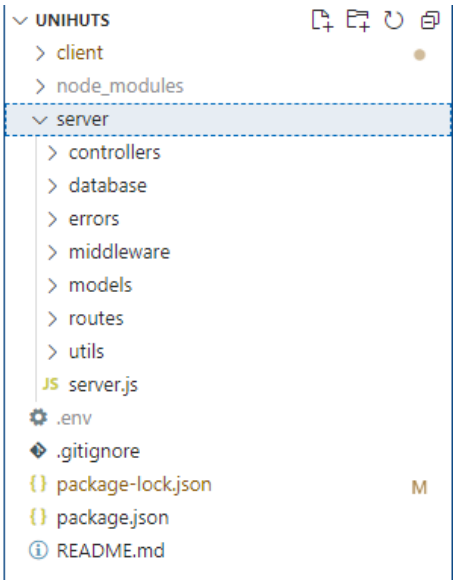
Parametri iz URL-a pohranjuju se uporabom *useParams() hook* iz React Router paketa. U ovom slučaju koriste se dva stanja pomoću *useState() hook*. Varijabla *room* koristi se za pohranu podataka o smještaju koja će se prikazati. Inicijalno je postavljeno na *null* jer se podaci o smještaju dohvaćaju asinkrono.

Unutar *useEffect() hook* postavljen je asinkroni zahtjev za dohvaćanje podataka o izabranom oglasu. Kroz asinkronu funkciju *getRoom()*, koristi se *axios* zahtjev kako bi se pozvao API */api/v1/public/room/${params.roomId}*. Funkcija *getRoom()* poziva se odmah nakon što se komponenta prikaže, zbog čega je *useEffect()* pozvan s praznim nizom ovisnosti ([ ]), što znači da će se izvršiti samo jednom, na montiranju komponente.

Osim standardnih ruta, React Router nudi mogućnost implementacije zaštićenih ruta koje zahtijevaju autentifikaciju. *ProtectedRoute* komponenta očekuje autentifikaciju korisnika i prikazuje sadržaj samo ako je korisnik autentificiran. To osigurava da samo prijavljeni korisnici mogu pristupiti određenim dijelovima aplikacije, u ovom slučaju riječ je o funkcionalnostima postavljanja novog oglasa za stambeno rješenje ili sustanara.

3.5. Razvoj pozadinske programske podrške

Express razvojni okvir, o čemu je više napisano u poglavlju 2.5., predstavlja jedan od najpopularnijih i najčešće korištenih Node razvojnih okvira za izgradnju pozadinske programske podrške. U ovom poglavlju opisana je primjena Express okvira u kreiranju ruta, obradi zahtjeva te generiranju odgovora. Dodano, opisane su osnove pristupa u rukovanju autentifikacijom i autorizacijom, osiguravajući sigurnost i zaštitu korisničkih podataka. Na slici 3.8. prikazana je krajnja struktura direktorija server.



Slika 3.8. Konačna struktura server direktorija unutar aplikacije.

Slijedi kratak opis sadržaja koji se nalazi u poddirektoriju server:

- *controllers* – sadrži kontrolere za različite funkcionalnosti servera
- *database* – sadrži datoteku *connect.js* odgovornu za uspostavu veze s bazom
- *errors* – sadrži datoteke koje definiraju prilagođene greške
- *middleware* – sadrži međuprograme koji se primjenjuju pri obradi zahtjeva
- *models* – sadrži definicije modela koji predstavljaju strukturu i logiku podataka u bazi
- *routes* – sadrži definicije putanja i metoda za svaku funkcionalnost aplikacije
- *utils* – korisne funkcije i alati koji se koriste na više mjesta u aplikaciji
- *server.js* – glavna datoteke pozadinske programske podrške.

3.5.1. Postavke i konfiguracija poslužitelja

Za uspješno postavljanje i konfiguraciju poslužitelja, odnosno pozadinske programske podrške, potrebno je inicijalizirati Express razvojni okvir pomoću *express()* funkcije, implementirati rute unutar aplikacije, implementirati odgovarajuće međuprograme poput *express.json()* koji omogućuje parsiranje tijela zahtjeva u JSON formatu, te naposljetku povezati poslužitelja i bazu podataka. Programski kod 3.7. prikazuje implementaciju prvog dijela *server.js* datoteke.

```
// * DATABASE
const connectDB = require("./database/connect");

const path = require("path");
const express = require("express");
const app = express();
const dotenv = require("dotenv");
dotenv.config();

require("express-async-errors");
const morgan = require("morgan");

const port = process.env.PORT || 5000;

// * ROUTERS
const authRouter = require("./routes/authRoutes");
const roomsRouter = require("./routes/roomsRoutes");
const roommatesRouter = require("./routes/roommatesRoutes");
const listingsRouter = require("./routes/listingsRoutes");

...
```

Programski kod 3.7. Konfiguracija poslužitelja unutar aplikacije.

Potrebno je uvesti module korištene od strane Express okvira - *path* se koristi za rad s putanjama datoteka i direktorija, *express* predstavlja sam Express paket koji omogućuje stvaranje i konfiguraciju poslužitelja, dok se *dotenv* koristi za učitavanje varijabli okruženja iz *.env* datoteke kako bi se sakrile osjetljive informacije poput administrativnih lozinki. Varijabla *port* postavlja vrijednost porta na kojem će Express server slušati. Ako je definirana varijabla okruženja *PORT*,

koristit će se ta vrijednost, u suprotnom će se koristiti podrazumijevana vrijednost *5000*. Osim navedenog, potrebno je uvesti datoteke koje sadrže definicije ruta za različite dijelove aplikacije, poput *authRouters*, *roomsRouters* i *roommatesRouters*. Nakon postavljenog osnovnog sloja Express poslužitelja, potrebno je konfigurirati srednje slojeve aplikacije i pokrenuti poslužitelja na već unaprijed definiranom portu. Implementacija navedenog vidljiva je u programskom kodu 3.8.

```
// * MIDDLEWARE
const notFoundMiddleware = require("./middleware/not-found");
const errorHandlerMiddleware = require("./middleware/error-handler");

if (process.env.NODE_ENV !== "production") {
	app.use(morgan("dev"));
}

app.use(express.json());
app.use(express.static(path.join(__dirname, "../client/build")));

app.use("/api/v1/public", listingsRouter);
app.use("/api/v1/auth", authRouter);
app.use("/api/v1/listing", roomsRouter);
app.use("/api/v1/listing", roommatesRouter);

app.get("*", (req, res) => {
	res.sendFile(__dirname, "../", "client", "build", "index.html");
});

app.use(notFoundMiddleware);
app.use(errorHandlerMiddleware);

const start = async () => {
	try {
		await connectDB(process.env.MONGO_URL);

		app.listen(port, () => {
			console.log(`Server is listening on port ${port}...`);
		});
	} catch (error) {
		console.log(error);
	}
};

start();
```

Programski kod 3.8. Konfiguracija srednjeg sloja poslužitelja unutar aplikacije.

3.5.2. Autentifikacija korisnika

Autentifikacija korisnika od ključne je važnosti u internet aplikacijama iz više razloga, a njen značaj proizlazi iz potrebe za zaštitom korisničkih podataka, osiguravanjem pristupa isključivo ovlaštenim korisnicima i održavanjem povjerenja između korisnika i internetske usluge. Dodatno, autentifikacija korisnika pruža mogućnost veće personalizacije i prilagođenog korisničkog iskustva svakom registriranom korisniku, uz preduvjet da aplikacija poštuje GDPR regulativu.

Programski kod 3.9. prikazuje *auth.js* međuprogram u kojem je implementirana autentifikacija korisnika pomoću JWT (eng. JSON Web Token) tokena. Funkcija *auth* predstavlja međuprogram

koji se primjenjuje na rute koje zahtijevaju autentifikaciju. Unutar funkcije se prvo provjerava prisutnost *Authorization* zaglavlja u zahtjevu, koje bi trebalo sadržavati JWT token, potom provjerava počinje li token s izrazom *Bearer*. Ako to nije slučaj, funkcija će baciti iznimku *UnAuthenticatedError*, što znači da korisnik nije autentificiran.

U bloku *try*, *jwt.verify()* metoda uspoređuje token s tajnim ključem za potpis kako bi provjerila njegovu valjanost i vremenski rok. Ako je token valjan, *userId* se izdvaja iz payloada i dodaje u *req.user*. Ako metoda baci iznimku, uhvaćena je u *catch* bloku i vraća *UnAuthenticatedError* iznimku, označavajući neuspjeh autentifikacije i nevaljan zahtjev.

```
const jwt = require("jsonwebtoken");
const { UnAuthenticatedError } = require("../errors/index");

const auth = async (req, res, next) => {
	const authHeader = req.headers.authorization;

	if (!authHeader || !authHeader.startsWith("Bearer")) {
		throw new UnAuthenticatedError("Authentication invalid!");
	}

	const token = authHeader.split(" ")[1];

	try {
		const payload = jwt.verify(token, process.env.JWT_SECRET);
		req.user = { userId: payload.userId };
		next();
	} catch (error) {
		throw new UnAuthenticatedError("Authentication invalid!");
	}
};

module.exports = auth;
```

Programski kod 3.9. Konfiguracija autentifikacije na poslužitelju.

3.5.3. Realizacija podatkovnog modela aplikacije

Realizacija podatkovnog modela na serveru provedena je pomoću Mongoose npm paketa. Ovaj paket omogućuje definiranje strukture podataka, odnosno sheme, koji se koriste za stvaranje, uređivanje, brisanje i pretraživanje dokumenata u MongoDB bazi. Sheme predstavljaju klase koje opisuju podatke, tipove i ograničenja podataka koje se mogu pohraniti u bazu. Programski kod 3.10. prikazuje implementaciju podatkovnog modela za korisnike. *userSchema* na vrlo jednostavan način opisuje strukturu i pravila za atribute koji su već definirani u MongoDB bazi podataka - *email*, *password*, *tosButton*, *firstName*, *lastName* i *userGender*. Svaki atribut ima propisani tip podatka i validaciju.

```
const userSchema = new mongoose.Schema({
	email: {
		type: String,
		required: [true, "Email address is required"],
		unique: true,
	},
	password: {
		type: String,
		required: [true, "Password is required"],
		select: false,
	},
	tosButton: {
		type: Boolean,
		required: [true, "TOS Agreement is required"],
	},

	firstName: {
		type: String,
		required: [true, "First name is required"],
		trim: true,
	},
	lastName: {
		type: String,
		required: [true, "Last name is required"],
		trim: true,
	},
	userGender: {
		type: String,
		required: false,
		default: "other",
	},
});
```

Programski kod 3.10. Implementacija podatkovnog modela za korisnika.

Dodatno, *userSchema* definira metodu za spremanje kriptiranih lozinki u bazu podataka, metodu za stvaranje jedinstvenog JWT tokena potpisanog tajnim ključem te metodu za usporedbu unesene lozinke s kriptiranom lozinkom. Implementacija navedenog vidljiva je u programskom kodu 3.11.

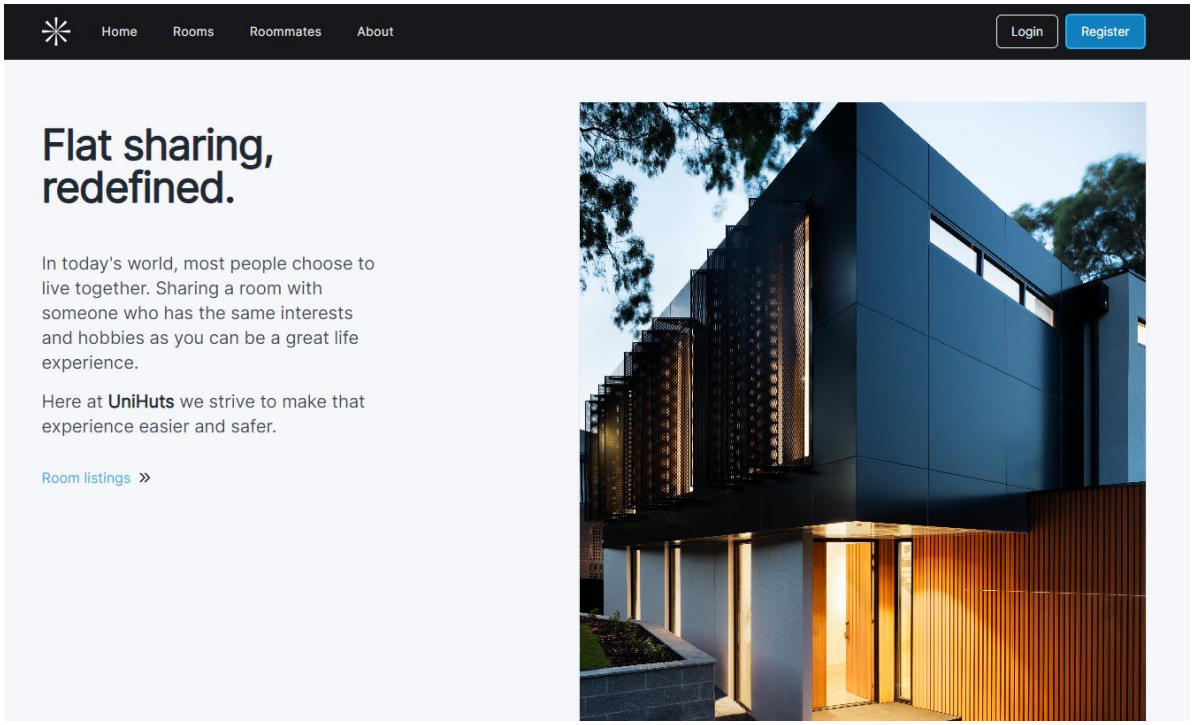
```
// * PASSWORD HASHING
userSchema.pre("save", async function () {
	if (!this.isModified("password")) return;

	const salt = await bcrypt.genSalt(10);
	this.password = await bcrypt.hash(this.password, salt);
});
// * JWT AUTH
userSchema.methods.createJWT = function () {
	return jwt.sign({ userId: this._id }, process.env.JWT_SECRET, {
		expiresIn: process.env.JWT_LIFETIME,
	});
};
// * PW COMPARING
userSchema.methods.comparePassword = async function (candidatePassword) {
	const isMatch = await bcrypt.compare(candidatePassword, this.password);
	return isMatch;
};

module.exports = mongoose.model("User", userSchema);
```

Programski kod 3.11. Autentifikacija korisnika vezana uz podatkovni model korisnika.

Zadnji korak uključuje *userSchema* pretvorbu u model *User*, što omogućuje manipulaciju korisničkim podacima i pristupanje njegovim metodama putem Mongoose operacija nad modelom. Pri prvom kreiranju dokumenta određenog modela u bazi podataka, mongoose kreira kolekciju s odgovarajućim imenom u množini, u ovom slučaju *users*.

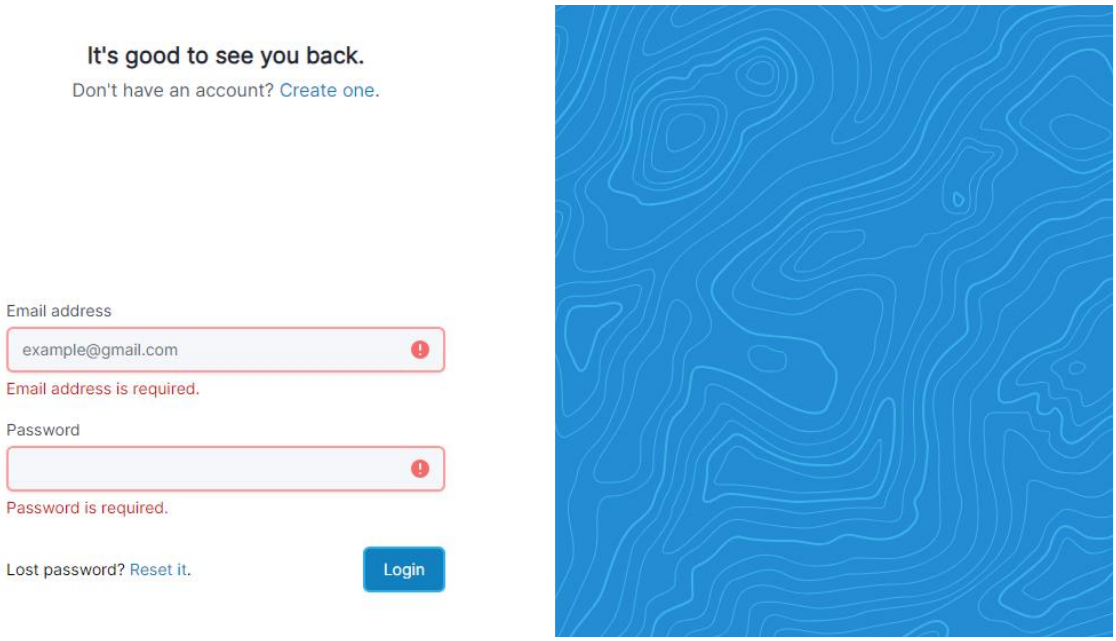
3.6. Konačna internet aplikacija

Prilikom prve posjete aplikaciji, korisnicima se na početnoj stranici prikazuju dvije glavne funkcionalnosti – objavljivanje oglasa za stambeno rješenje i objavljivanje oglasa za sustanara. Na slici 3.9. vidljiva je početna stranica i navigacijska traka koja korisnicima omogućuje pregledavanje javno dostupnih stambenih rješenja i aktivni oglasi korisnika koji traže sustanare.



Slika 3.9. Početna stranica aplikacije.

Klikom na *Login* u gornjem desnom kutu, korisnika se preusmjerava na stranicu za prijavu. Neregistriranim korisnicima se nudi mogućnost kreiranja računa. Slika 3.10. prikazuje stranicu za prijavu u aplikaciju, gdje se može vidjeti i validacija polja implementirana yup npm paketom. Programski kod 3.12. prikazuje *LoginShema* datoteku za provjeru ispravnosti podataka koji se unose u obrazac za prijavu.

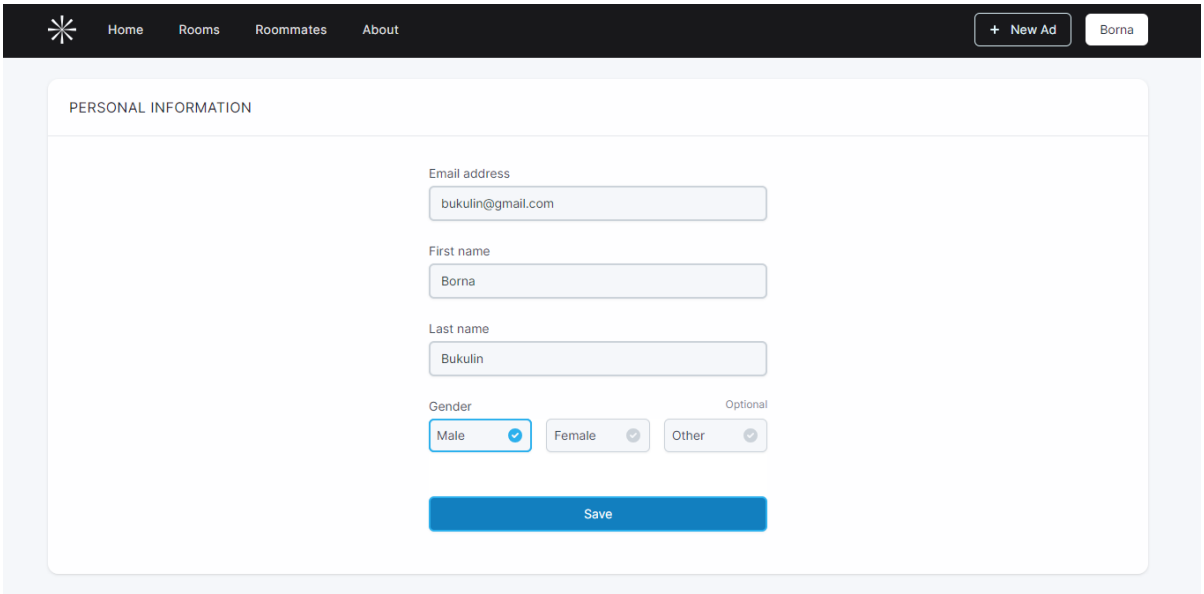


Slika 3.10. Stranica za prijavu na oglasnik.

```
import * as Yup from "yup";

const LoginSchema = Yup.object({
       email: Yup.string()
              .email("Invalid email")
              .required("Email address is required."),
       password: Yup.string().required("Password is required."),
});

export default LoginSchema;
```

Programski kod 3.12. LoginShema dokument.

Nakon uspješne prijave ili registracije na oglasnik, korisniku će se umjesto poveznica *Login* i *Register* u gornjem desnom kutu aplikacije prikazati padajući izbornik u kojem ima mogućnost pregledati svoje oglase, odjaviti se iz aplikacije ili pak promijeniti podatke svog korisničkog računa. Slika 3.11. prikazuje zaslon postavki korisničkog računa. Implementacija funkcionalnosti izmjene vlastitih korisničkih podataka prikazana je u programskom kodu 3.13. Kod predstavlja dio *UpdateSettingsForm* komponente koja koristi *useAppContext() hook* čiji je zadatak dohvatiti stanje o trenutno prijavljenom korisniku i funkciju za ažuriranje korisničkih podataka *updateUser*. Varijabla *initialValues* omogućuju ispunjavanje polja obrasca podatcima trenutno prijavljenog korisnika.

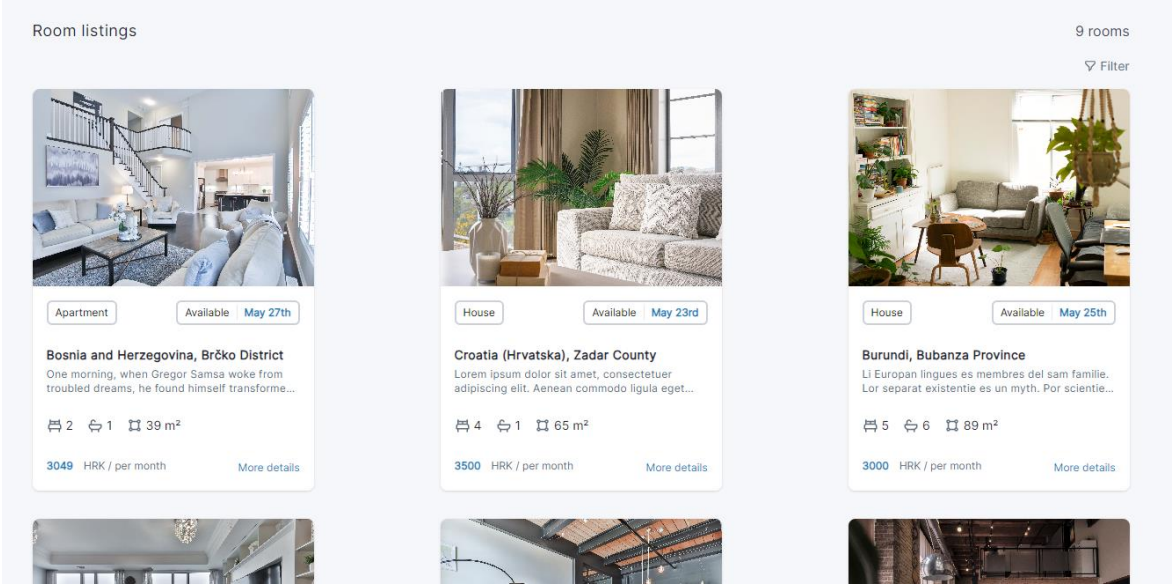


Slika 3.11. Postavke korisničkog računa.

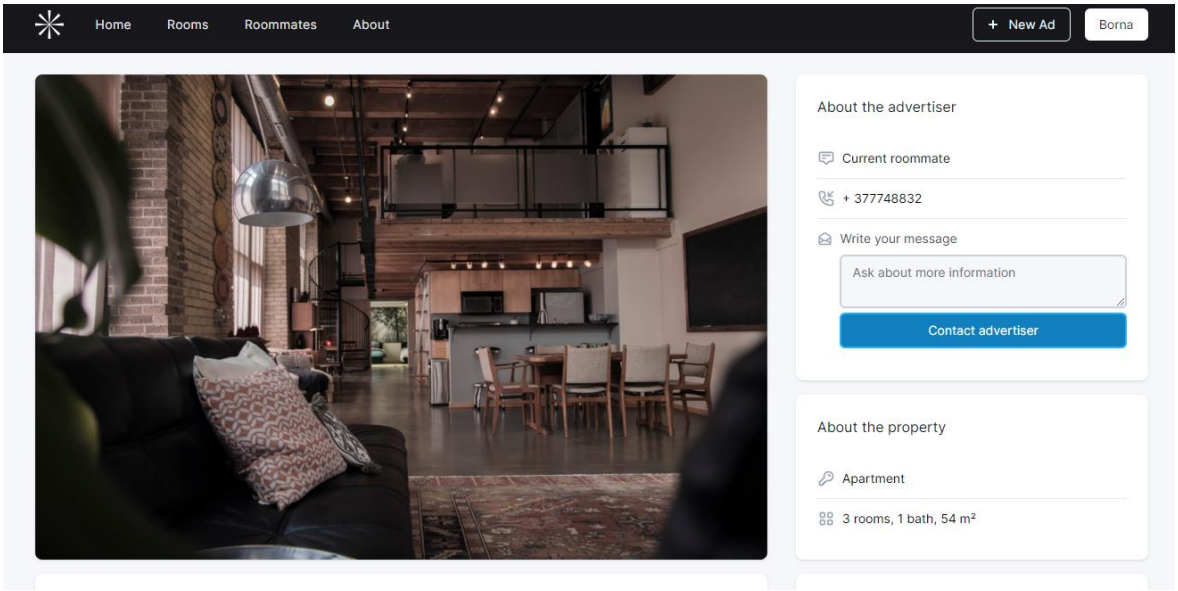
```
function UpdateSettingsForm() {
	const { user, showAlert, displayAlert, updateUser, isLoading } = useAppContext();

	return (
	<div>
		<div>
			<div>
				<p> Personal information </p>
			</div>
			<div>
				<div>
				<Formik
					initialValues={{
						email: user.email,
						firstName: user.firstName,
						lastName: user.lastName,
						userGender: user.userGender,
					}}
					onSubmit={({email, firstName, lastName, userGender}) => {
						if (!email || !firstName || !lastName) {
								displayAlert();
								return;
							}
						updateUser({email,firstName,lastName,userGender});
					}}>
				<Form>
				<TextInput label="Email address" name="email" type="email"/>

				<TextInput label="First name" name="firstName" type="text" />

				<TextInput label="Last name" name="lastName" type="text" />
```

Programski kod 3.13.Dio implementacije UpdateSettingsForm komponente.

Klikom na *Rooms* poveznicu u navigacijskoj traci, korisnik biva preusmjeren na listu trenutno aktivnih oglasa stambenih rješenja. Korisnik ima uvid u ukupan broj oglasa i mogućnost filtriranja oglasa s ciljem bržeg pronalaska stambenog rješenja prema vlastitim afinitetima. Slika 3.12. prikazuje stranicu na kojoj su vidljivi trenutno aktivni oglasi. Klikom na *More details* poveznicu sadržanu u svakoj kartici pojedinačnog oglasa, korisnik biva preusmjeren na detaljnu stranicu određenog oglasa, kako je i vidljivo na slici 3.13.

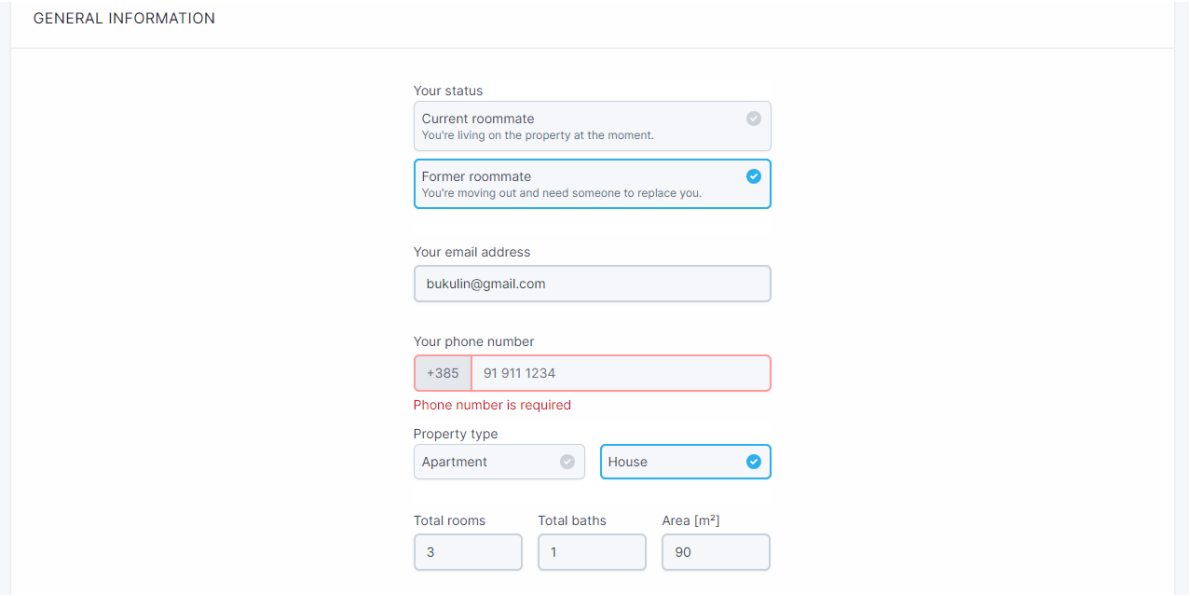


Slika 3.12. Lista trenutno aktivnih oglasa.



Slika 3.13. Detaljni prikaz određenog oglasa.

Klikom na *New Ad* u gornjem desnom kutu navigacijske trake korisnik će imati na izbor kreirati novi oglas za stambeno rješenje ili sustanara. Slika 3.14. prikazuje ispunjavanje općenitih informacija kod objavljivanja stambenog rješenja. Veliku ulogu u dobrom korisničkom iskustvu ima i responzivni dizajn. Aplikacija je u potpunosti prilagođena za sve dimenzije zaslona, omogućavajući na taj način korisnicima objavljivanje oglasa sa osobnih računala, pametnih telefona ili tablet uređaja.



Slika 3.14. Ispunjavanje općenitih informacija na računalnoj verziji aplikacije.

4. ZAKLJUČAK

Završnim radom demonstrirana je implementacija internet aplikacije fokusirane na pretraživanje i oglašavanje smještaja, koristeći MERN arhitekturu. Kroz integraciju MongoDB baze podataka, Express razvojnog okvira, React JavaScript biblioteke i Node.js razvojne okoline, ostvareno je jednostavno skalabilno rješenje. Navedeni su razlozi odabira MERN arhitekture, kao i prednosti svake od tehnologija i alata korištenih pri izradi internet aplikacije. Proučena je važnost modeliranja podatkovnog modela u NoSQL bazama podataka. Prikazana je implementacija upravljanja globalnim stanjem unutar React aplikacija korištenjem React Context API. Na primjeru je objašnjena važnost implementacije zaštićenih ruta za kontrolu pristupa internet aplikaciji. Postavljene su temelje značajke autentifikacije korisnika unutar internet aplikacije i realizacija podatkovnog modela. U konačnici, snimke zaslona internet aplikacije prikazuju rezultat implementacijskog dijela završnog rada.

LITERATURA

[1] E. A. Scott, *SPA design and architecture: understanding single-page web applications*. Shelter Island, NY: Manning, 2016.

[2] HTML Standard, [Online]. Dostupno na: https://html.spec.whatwg.org/ [pristupljeno 26. kolovoz 2023.].

[3] N. Rappin, *Modern CSS with Tailwind: flexible styling without the fuss*. u Pragmatic exPress. Raleigh: The Pragmatic Bookshelf, 2021.

[4] Most used languages among software developers globally 2023, *Statista*, [Online.] Dostupno na: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ [26. kolovoz 2023.].

[5] S. Bradshaw, K. Chodorow, i E. Brazil, *MongoDB: the definitive guide: powerful and scalable data storage*, Third edition. Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc, 2020.

[6] E. Hahn, *Express in action: writing, building, and testing Node.js applications*. Shelter Island, NY: Manning Publications, 2016.

[7] Stack Overflow Developer Survey 2023, *Stack Overflow*, [Online]. Dostupno na: https://survey.stackoverflow.co/2023/ [26. kolovoz 2023.].

[8] React, [Online]. Dostupno na: https://react.dev/ [26. kolovoz 2023.].

[9] M. Sharma, Understanding Functional Components Vs. Class Components in React, *Scaler Topics*, [Online]. Dostupno na: https://www.scaler.com/topics/react/react-functional-vs-class-components/ [26. kolovoz 2023.].

[10] Tailwind CSS - Rapidly build modern websites without ever leaving your HTML, [Online]. Dostupno na: https://tailwindcss.com/ [27. lipanj 2022.].

[11] Font Size - Tailwind CSS, [Online]. Dostupno na: https://tailwindcss.com/docs/font-size [26. kolovoz 2023.].

[12] BSON (Binary JSON): Specification, [Online]. Dostupno na: https://bsonspec.org/spec.html [26. kolovoz 2023.].

[13] 15 Best Web Development IDE You Should Pick in 2023 [Updated], *Hackr.io*, [Online]. Dostupno na: https://hackr.io/blog/web-development-ide [26. kolovoz 2023.].

[14] MongoDB Atlas Database | Multi-Cloud Database Service, *MongoDB*, [Online]. Dostupno na: https://www.mongodb.com/atlas/database [28. lipanj 2022.].

[15] MongoDB Compass, *MongoDB*, [Online]. Dostupno na: https://www.mongodb.com/products/compass [28. lipanj 2022.].

[16] Passing Data Deeply with Context – React, [Online]. Dostupno na: https://react.dev/learn/passing-data-deeply-with-context [26. kolovoz 2023.].

SAŽETAK

Završni rad opisuje izradu internet aplikacije koja omogućuje korisnicima jednostavno korisničko iskustvo kroz složeni proces pronalaska cjenovno prihvatljivog stambenog rješenja ili odgovarajućeg sustanara. Opisane su glavne značajke i prednosti MERN arhitekture korištene pri izradi rada. Ukratko su navedene razlike između SQL i NoSQL baza podataka. Na primjeru MongoDB baze podataka istaknuta je važnost modeliranja NoSQL baza podataka. Istražena je razlika između Express razvojnog okvira i Node razvojne okoline. Opisane su React komponente unutar virtualnog DOM-a. Primjeri u radu demonstriraju implementaciju globalnog stanja React aplikacije koristeći React Context API, kao i način upravljanja rutama aplikacije. Rad ističe korake postavljanja i konfiguracije Express pozadinske programske podrške. Demonstrirana je implementacija procesa autentifikacije korisnika te realizacija podatkovnog modela aplikacije. U konačnici, rezultati implementacije internet aplikacije prikazani su snimkama zaslona.

Ključne riječi: Express, MERN, NoSQL, React Context API.

ABSTRACT

Internet application for accommodation search and advertising

The thesis showcases the development of an internet application aimed at providing users with a simple user experience during the complex process of searching for an affordable housing solution or a suitable roommate. It discusses the main features and advantages of the MERN architecture used in the application's development, briefly outlining the differences between SQL and NoSQL databases. The importance of NoSQL database modeling is emphasized using MongoDB as an example. Additionally, the thesis clarifies the differences between Express library and Node framework. It explores React components within the virtual DOM and offers concrete examples that demonstrate how to implement global state management in a React application using the React Context API, while also providing insights into managing application routes. The thesis also highlights the steps involved in setting up and configuring the Express backend. It demonstrates the implementation of the user authentication process and the realization of the application data model. Lastly, the thesis showcases the results of the internet application's implementation with accompanying screenshots.

Keywords: Express, MERN, NoSQL, React Context API

ŽIVOTOPIS

Borna Bukulin rođen je 29. ožujka 2000. godine u Vinkovcima. Nakon završenog općeg smjera Gimnazije Matije Antuna Reljkovića, godine 2019. upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek pri Sveučilištu Josipa Jurja Strossmayera u Osijeku. Na samom početku studija postaje aktivan član sveučilišnog IEEE studentskog ogranka, gdje je godinu dana obavljao dužnost potpredsjednika, a potom i predsjednika ogranka. Za vrijeme treće godine studija pridružuje se tvrtki CROZ d.o.o sa sjedištem u Zagrebu, gdje radi na poziciji Product Designer & Analyst.

Borna Bukulin