



UNIVERZITET U NOVOM SADU  
FAKULTET TEHNIČKIH NAUKA



---

Vladimir Cvetanović


**REST klijent i RESTFul servis  
u web aplikaciji  
za rezervaciju smeštaja**

DIPLOMSKI RAD

- Osnovne akademske studije –

Novi Sad, 2019



	UNIVERZITET U NOVOM SADU • <b>FAKULTET TEHNIČKIH NAUKA</b> 21000 NOVI SAD, Trg Dositeja Obradovića 6	Datum:
	<b>ZADATAK ZA IZRADU ZAVRŠNOG (BACHELOR) RADA</b>	List/Listova:
		1/1

(Podatke unosi predmetni nastavnik - mentor)

Vrsta studija:	<input type="checkbox"/> Osnovne akademske studije
Studijski program:	<b>Računarstvo i automatika</b>
Rukovodilac studijskog programa:	<b>dr Milan Vidaković</b>

Student:	<b>Vladimir Cvetanović</b>	Broj indeksa:	<b>RA142/2015</b>
Oblast	<b>XML i web servisi</b>		
Mentor:	<b>Miroslav Zarić</b>		

NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA IZDAJE SE ZADATAK ZA DIPLOMSKI (Bachelor) RAD, SA SLEDEĆIM ELEMENTIMA:

- problem – tema radaпроблем – тема рада;
- način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna;
- literatura;

## NASLOV DIPLOMSKOG (BACHELOR) RADA:

<b>REST klijent i RESTFul servis u web aplikaciji za rezervaciju smeštaja</b>
---

## TEKST ZADATKA:

Kandidat treba da prouči koncepte servisno orijentisanih softverskih arhitektura, web servisa i RESTFul servisa. Analizirati razvojne okvire i alate za razvoj aplikacija koje koriste REST servise. U praktičnom delu prikazati implementaciju jednog ovakvog servisa. Diskutovati rešenje.

Rukovodilac studijskog programa:	Mentor rada:

Primerak za: ☐ - Studenta; ☐ - Mentora





UNIVERZITET U NOVOM SADU • FAKULTET TEHNIČKIH NAUKA  
21000 NOVI SAD, Trg Dositeja Obradovića 6

## KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, <b>RBR</b> :		
Identifikacioni broj, <b>IBR</b> :		
Tip dokumentacije, <b>TD</b> :	monografska publikacija	
Tip zapisa, <b>TZ</b> :	tekstualni štampani dokument	
Vrsta rada, <b>VR</b> :	diplomski rad	
Autor, <b>AU</b> :	Vladimir Cvetanović	
Mentor, <b>MN</b> :	dr Miroslav Zarić, FTN Novi Sad	
Naslov rada, <b>NR</b> :	REST klijent i RESTful servis u web aplikaciji za rezervaciju smeštaja	
Jezik publikacije, <b>JP</b> :	srpski	
Jezik izvoda, <b>JL</b> :	srpski / engleski	
Zemlja publikovanja, <b>ZP</b> :	Srbija	
Uže geografsko područje, <b>UGP</b> :	Vojvodina	
Godina, <b>GO</b> :	2019	
Izdavač, <b>IZ</b> :	autorski reprint	
Mesto i adresa, <b>MA</b> :	Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6	
Fizički opis rada, <b>FO</b> :	br. poglavlja: 7 / stranica: 49 / citata: 0 / slika: 21	
Naučna oblast, <b>NO</b> :	Računarske nauke i informatika	
Naučna disciplina, <b>ND</b> :	XML i web servisi	
Predmetna odrednica / ključne reči, <b>PO</b> :	Web servisi, servisno orijentisane arhitekture, RESTful	
<b>UDK</b>		
Čuva se, <b>ČU</b> :	Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad	
Važna napomena, <b>VN</b> :		
Izvod, <b>IZ</b> :	U radu je prikazan model podataka, kao i primenjene tehnologije potrebne za implementaciju sistema za rezervaciju smeštaja. Detaljno je opisana implementacija RESTful web servisa u komunikaciji između korisničke aplikacije i glavne aplikacije za potrebe pretraživanja i rezervisanja smeštaja.  Prikazana implementacija RESTful web servisa omogućava jednostavnu i efikasnu realizaciju funkcionalnosti koja se odnosi na pretraživanje i rezervisanje smeštaja.	
Datum prihvatanja teme, <b>DP</b> :		
Datum odbrane, <b>DO</b> :		
Članovi komisije, <b>KO</b> :		
Predsednik:	dr Gordana Milosavljević, vanredni profesor, FTN Novi Sad	
Član:	dr Goran Sladić, vanredni profesor, FTN Novi Sad	Potpis mentora:
Mentor:	dr Miroslav Zarić, vanredni profesor, FTN Novi Sad	





UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES  
21000 NOVI SAD, Trg Dositeja Obradovića 6

## KEY WORDS DOCUMENTATION

Accession number, <b>ANO</b> :			
Identification number, <b>INO</b> :			
Document type, <b>DT</b> :	monographic publication		
Type of record, <b>TR</b> :	textual material		
Contents code, <b>CC</b> :	BSc thesis		
Author, <b>AU</b> :	Vladimir Cvetanović		
Mentor, <b>MN</b> :	Miroslav Zarić, PhD assoc. prof., FTN Novi Sad		
Title, <b>TI</b> :	REST Client and RESTful Service In Online Booking Application		
Language of text, <b>LT</b> :	Serbian		
Language of abstract, <b>LA</b> :	serbian / english		
Country of publication, <b>CP</b> :	Serbia		
Locality of publication, <b>LP</b> :	Vojvodina		
Publication year, <b>PY</b> :	2019		
Publisher, <b>PB</b> :	author's reprint		
Publication place, <b>PP</b> :	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6		
Physical description, <b>PD</b> :	no. of chapters: 7 / pages: 49 / citations: 0 / images: 21		
Scientific field, <b>SF</b> :	Computer science		
Scientific discipline, <b>ND</b> :	XML and Web Services		
Subject / Keywords, <b>S/KW</b> :	Web services, service oriented architecture, RESTful		
<b>UDC</b>			
Holding data, <b>HD</b> :	Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad		
Note, <b>N</b> :			
Abstract, <b>AB</b> :			
Accepted by sci. board on, <b>ASB</b> :	<p>This paper the data model as well as the technologies used to implement the accommodation booking system. The implementation of the RESTful web service in communication between the user application and the main application for searching and booking accommodation is described in detail.</p> <p>The presented implementation of the RESTful web service enables the simple and efficient realization of the functionality related to searching and booking accommodation.</p>		
Defended on, <b>DE</b> :			
Defense board, <b>DB</b> :			
	President:	Gordana Milosavljević, PhD, assoc. prof., FTN Novi Sad	
	Member:	Goran Sladić, PhD, assoc. prof., FTN Novi Sad	Mentor's signature
	Mentor:	Miroslav Zarić, PhD, assoc. prof., FTN Novi Sad	





# Spisak korišćenih skraćenica

Skraćenica – pun naziv

REST – Representational State Transfer  
API – Application programming interface  
JSON – JavaScript Object Notation  
XML – Extensible Markup Language  
WSDL – Web Services Description Language  
UDDI – Universal Description, Discovery and Integration  
SOAP - Simple Object Access Protocol  
HTTP – Hyper Text Transfer Protocol  
HTML – Hyper Text Markup Language  
TCP – Transmission Control Protocol  
IP – Internet Protocol  
URI – Uniformed Resources Identifier  
CRUD – Create Read Update Delete  
DOM – Document Object Model  
MVC – Model View Controller  
SQL – Structured Query Language

# Sadržaj

Sadržaj .....	10
1. Uvod.....	11
2. Teorijske osnove .....	12
2.1. Web servisi.....	12
2.2. Tipovi web servisa .....	12
2.2.1 SOAP.....	12
2.2.2 REST.....	14
2.3 JSON.....	16
2.4 XML .....	17
2.5 WSDL .....	19
2.6 UDDI .....	19
2.6.1 Bele stranice .....	19
2.6.2 Žute stranice .....	20
2.6.3 Zelene stranice.....	20
3. Opis arhitekture / model sistema.....	21
3.1. Arhitektura sistema .....	21
3.1.1 Spring Boot .....	22
3.1.2 Angular framework.....	24
3.1.3 MySQL .....	25
3.1.4 Google Cloud .....	26
3.2 Model sistema.....	27
4. Implementacija .....	31
4.2 Pretraga smeštaja .....	31
4.2 Rezervacija smeštaja.....	35
5. Zaključak.....	41
6. Literatura .....	42
7. Biografija.....	44

# 1. Uvod

Predmet diplomskog rada je realizacija informacionog sistema za rezervaciju smeštaja putem interneta. Poseban akcenat stavlja se na opisu implementacije komunikacije između glavne aplikacije (*back-end*) i klijentske aplikacije putem RESTful web servisa.

U 2. poglavlju je dat opis teorijskih osnova potrebnih za razumevanja koncepta web servisa. Pored web servisa, opisani su pojmovi kao što su JSON, XML, WSDL i UDDI.

U poglavlju 3. dat je opis arhitekture i model podataka sistema. Glavna aplikacija je implementirana putem Spring Boot razvojnog okvira (engl. *framework*), sadrži glavnu bazu podataka, svu potrebnu poslovnu logiku aplikacije i neophodne web servise. Klijentska aplikacija je implementirana putem AngularJS *framework*-a za razvoj dinamičkih web aplikacija. Ova aplikacija omogućuje korisnicima da komunicira sa glavnom aplikacijom i da obavlja funkcionalnosti koja ona nudi.

U poglavlju 4. opisani su i prikazani svi delovi implementacije koji se odnose na komunikaciju klijentske aplikacije i glavne aplikacije, koji se odnose na rezervaciju smeštaja. Rezervacija smeštaja je postupak koji obuhvata: pretragu smeštaja i pravljenje same rezervacije. Pretraga smeštaja može biti osnovna i proširena. Za osnovnu se mogu pretražiti smeštajne jedinice po mestu, datumu početka, datumu završetka i po broju gostiju. Za proširenu se dobijaju dodatne opcije za pretragu poput izbora tipa smeštaja (hotel, apartman i sl.), kategorije smeštaja, distance od grada i dodatnih usluga smeštaja (wi-fi, parking, polu pansino, pun pansion i sl.). Ostali servisi koje aplikacija nudi su realizovani po istom principu kao i servis za rezervaciju smeštaja.

U poglavlju 5. data su zaključna razmatranja.

## 2. Teorijske osnove

Za realizaciju ovog projekta korišćene su različite tehnologije i metodologije. Osnovu ovog projekta čine: web servisi i RESTful servisi. Razmena podataka između glavne aplikacije i agentske aplikacije vrši se putem SOAP baziranih web servisa gde se podaci šalju u vidu XML dokumenta. Između glavne aplikacije i klijentske aplikacije razmena podataka je realizovana putem RESTful servisa gde se podaci šalju u vidu JSON dokumenta.

### 2.1. Web servisi

Web servis je kolekcija otvorenih protokola i standarda koji se koriste za razmenu podataka između aplikacija i sistema. Softverske aplikacije koje su napisane u raznim programskim jezicima i koje su pokrenute na različitim platformama mogu da koriste web servise da razmenjuju podatke putem računarskih mreža. Ovo je moguće zahvaljujući otvorenim standardima i protokolima poput TCP/IP, HTTP, XML i HTML. [1]

Komponente koje čine web servis:

1. XML – koristi se za reprezentaciju i razmenu podataka
2. SOAP – standardni način komunikacije
3. UDDI – mehanizam koji služi da se registruje i locira servis
4. WSDL – standardni jezik za opis servisa

### 2.2. Tipovi web servisa

#### 2.2.1 SOAP

**SOAP (Simple Object Access Protocol)** je standardan način komunikacije između dva web servisa koji se bazira na korišćenju XML jezika za definiciju podataka koji se prenose putem mreže. SOAP podržava dva stila za komunikaciju:

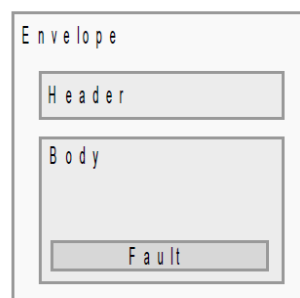
1. **RPC (Remote Procedure Call)** – Protokol koji se koristi da se pozove funkcija koja se nalazi na udaljenom servisu. Koristi klijent/server model za komunikaciju. Tipičan je za sinhronu komunikaciju.

2. Dokument orijentisani – Podržava komunikaciju između sistema putem slanja (struktuiranih) poruka. Tipičan je za asinhronu komunikaciju.

Za razmenu poruka SOAP se oslanja najčešće na HTTP protokol (može i na SMTP). Razlog zbog kog se koristi HTTP protokol kao podrazumevani jeste to što je HTTP firewall-friendly.

Zbog te osobine protokola omogućena je lakša komunikacija putem interneta i globalna dostupnost web servisa. Komunikacija putem SOAP-a je bazirana na razmeni SOAP poruka. SOAP poruke se formiraju kao XML dokumenti i sadrže sledeće elemente:

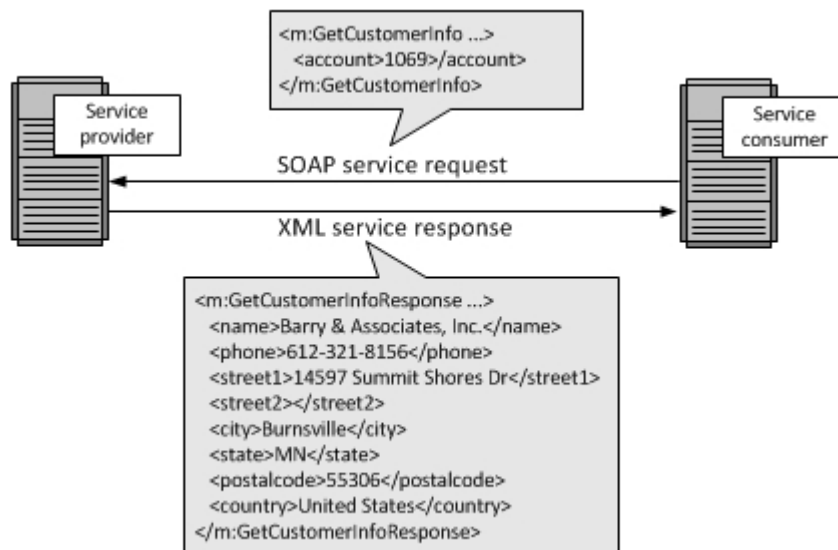
```
<?xml version="1.0"?>
<soap:Envelope>
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```



XML i web servisi 2008.

**Slika 2.0:** Struktura SOAP poruke [2]

**Envelope (korenski element)** obuhvata celu SOAP poruku. **Header** sadrži podatke koji opisuju kontekst u kome se šalje poruka ili uputstva za posrednike u komunikaciji između krajnjih korisnika. Ovi podaci ne predstavljaju samu poruku, već pomoćne podatke koji utiču na način obrade poruke. **Body** sadrži konkretan SOAP zahtev ili odgovor i koristi se kod svih vrsta web servisa. **Fault** je opcioni podlement, sadrži podatke o nastalim greškama namenjene klijentu. [3]



**Slika 2.1:** Primer SOAP zahteva i odgovora [4]

## 2.2.2 REST

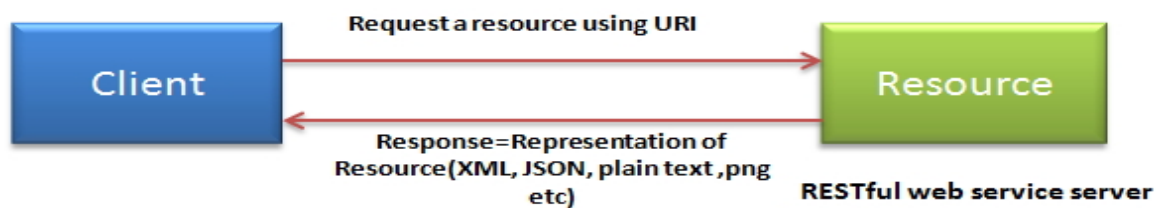
REST je stil arhitekture koji definiše ograničenja, koja ako se primene na web servise daju poželjne osobine poput performanse, skalabilnosti i mogućnosti modifikovanja. U REST arhitekturi, podaci i funkcionalnosti se smatraju resursima kojima se može pristupiti putem URI-a, tipično linkovi na internetu. Definiše klijent/server arhitekturu i dizajniran je da koristi HTTP protokol kao način komunikacije. Resursima se manipuliše putem CRUD (**Create Read Update Delete**) operacija koje se oslanjaju na HTTP metode: GET, PUT, POST, DELETE. Klijenti i serveri razmenjuju reprezentacije resursa korišćenjem standardizovanih protokola i interfejsa. Resursi se mogu predstaviti u JSON formatu (najčešće korišćen) ili XML formatu. [5]

Ograničenja koje specificira REST arhitektura :

1. Klijent-server separacija – Najfundamentalniji zahtev REST arhitekture. Server i klijent su nezavisni. Interakcija između njih je samo putem zahteva, koje šalje klijent, i odgovora, koji server šalje klijentu nakon što obradi zahtev.
2. *Stateless* (bez stanja) – Ovaj koncept podrazumeva da klijent mora da pošalje sve potrebne informacije serveru. Ovo je potrebno kako bi server mogao na odgovarajući

način da odgovori klijentu. Server ne treba da zadržava bilo koje informacije između zahteva koje klijent šalje.

3. *Cache* (keš) – Koncept koji rešava problem *stateless*-a. Pošto je komunikacija između servera i klijenta nezavisna, moguće je da klijent pošalje zahtev koji je već u prošlosti uputio serveru. Ovo povećava saobraćaj u mreži. *Cache* koncept rešava ovaj problem tako što čuva takve zahteve koji su već poslali serveru. Ako se takav zahtev opet pošalje, onda se pristupa cache-u i dobija se potrebna informacija umesto da se opet šalje zahtev serveru.
4. Slojevit sistem – Koncept u kom je moguće dodati bilo koj dodatni sloj između klijenta i stvarnog servera koji je domaćin RESTful web servisa. Uvođenje svakog dodatnog sloja mora biti transparentno da ne bi poremetilo interakciju između klijenta i servera.
5. Uniformni interfejs – Resursima se manipuliše putem četiri operacije: kreiranje, čitanje, ažuriranje i brisanje. Ovo je omogućeno putem HTTP zahteva: GET, PUT, POST, DELETE. GET dobavlja trenutno stanje resursa, PUT menja stanje ili ažurira stanje resursa, POST kreira novi resurs na serveru, DELETE briše resurs.



Slika 2.2: Primer REST zahteva [7]

REST definiše određene standarde i smernice koje se tiču korišćenja HTTP metoda za određene CRUD operacije (smernice je tehnički moguće kršiti ali se ne preporučuje):

- GET – Koristiti GET metodu jedino za dobavljanje informacija o resursu, nikako za promenu resursa. Kako GET zahtevi ne menjaju stanje resursa, oni se smatraju **bezbednim zahtevima**. Pored toga GET zahtevi treba da budu **idempotentni**, što znači da upućivanje više identičnih zahteva mora da proizvede isti rezultat svaki put. Za bilo koji GET zahtev, ako je resurs pronađen na serveru, onda server mora da vrati statusni kod **200 (OK)** zajedno sa resursom koji se nalazi u telu odgovora i uglavnom je u JSON formatu. Ukoliko resur nije pronađen na serveru, onda mora da vrati statusni kod **404 (NOT FOUND)**. Slično tome, ukoliko se utvrdi da sam GET zahtev nije dobro formiran od strane klijenta, server će vratiti statusni kod **400 (BAD REQUEST)**.

- **POST** – Koristiti POST metodu za kreiranje novog resursa. Govoreći strogo u smislu REST-a, POST metoda se koristi da bi se stvorio novi resurs u kolekciji resursa. U idealnom slučaju, ako je resurs uspešno kreiran, odgovor od strane servera treba da bude statusni kod **201 (CREATED)** i da u telu odgovora bude entitet koji je kreiran. Moguće je da POST metoda ne rezultira resursom koji može biti identifikovan pomoću URI-a. U ovom slučaju je odgovarajući odgovor statusni kod **204 (NO CONTENT)** ili **200 (OK)**. POST metoda nije **bezbedna** ni **idempotentna**, slanje dva identična zahteva rezultiraće u dva različita resursa koji sadrži iste informacije (osim identifikatora resursa).
- **PUT** – Koristiti PUT metodu primarno za ažuriranje resursa. Ukoliko je novi resurs kreiran putem PUT metode, server mora da obavesti o tome klijenta sa statusnim kodom **201 (CREATED)**. Ako je postojeći resurs modifikovan, potrebno je da se pošalje odgovor sa statusnim kodom **200 (OK)** ili **204 (NO CONTENT)** kako bi se ukazalo na uspešno izvršavanje zahteva.
- **DELETE** – Koristiti DELETE metodu za brisanje resursa koji je identifikovan putem URI zahteva. Uspešan odgovor DELETE zahteva treba da bude statusni kod **200 (OK)** ako odgovor uključuje entitet koji opisuje status, **202 (ACCEPTED)** ako je akcija stavljena u red čekanja ili **204 (NO CONTENT)** ako je akcija izvršena a ne uključuje entitet. DELETE metoda je **idempotentna**. Ako se resurs obriše, on se uklanja iz kolekcije resursa. Ponovno pozivanje DELETE metode neće promeniti ishod, samo je potrebno da se kao odgovor pošalje statusni kod **404 (NOT FOUND)** jer je već uklonjen.

## 2.3 JSON

JSON je jednostavan format za čuvanje i razmenu podataka. Često se koristi kada se podaci šalju sa servera na web stranicu. JSON je “samoopisiv” i lako razumljiv. Format JSON-a je sintaksno identičan kodu kojim se kreiraju JavaScript objekti. Zbog ove sličnosti, JavaScript program može lako da konvertuje JSON u JavaScript objekat. Prednost JSON-a je u tome što je njegov format tekstualan, može lako da se koristi od strane bilo kog programskog jezika.

Podaci u JSON-u se pišu kao parovi “*naziv:vrednost*” i odvojeni su zarezom. Podaci mogu biti: string-ovi, brojevi, objekti, nizovi, 16oolean vrednosti (0 ili 1). String-ovi se pišu pod znacima navoda. Brojevi mogu biti celi ili razlomljeni. Objekti se definišu unutar vitičastih zagrada i sadrže više parova “*naziv:vrednost*”. Nizovi i kolekcije pišu se unutar kockastih zagrada. Kao i u JavaScript-u, niz može da se sastoji od objekata. [8]



Primer izgleda jednog JSON fajla:

```
{  
  "firstname": "Michael",  
  "lastname": "Hopkins",  
  "age": 30,  
  "address": {  
    "street": "34 Fifth Street",  
    "city": "New York",  
    "state": "NY"  
  }  
}
```

Kod 2.0

## 2.4 XML

XML, proširivi jezik za označavanje (**eXtensible Markup Language**), je softverski i hardverski nezavisan alat za skladištenje i prenos podataka. Predstavlja metajezik za opis elemenata koje korisnik sam definiše. Postoje tri bitne karakteristike XML-a koje ga čine korisnim u raznim sistemima:

- Proširivost – XML omogućava pravljenje ličnih elemenata i samoopisivih tagova.
- XML nosi podatke, ne služi za prikaz – XML omogućava da se podaci čuvaju bez obzira na to kako će biti prezentovani.
- XML je javni standard – XML je razvijen od strane organizacije koja se zove **World Wide Web Consortium (W3C)** i dostupan je kao otvoren standard.

Osnovna jedinica XML-a je tag. Tagovi identifikuju podatke, oni ne specificiraju kako će oni biti prikazani. XML ne koristi predefinisane tagove za označavanje elemenata. Naziv taga je ograđen sa simbolima "<" i ">", što se može videti u sledećem primeru:

```
<naziv_taga>
```

Kod 2.1

Jedan XML dokument je sastavljen od više elemenata. Sadržaj elementa se nalazi između početnog i zatvarajućeg taga. Svaki element se završava zatvarajućim tagom.

```
<naziv_taga> ... </naziv_taga>
```

Kod 2.2

Nije dozvoljeno ugnježdavanje elemenata - elementi unutar dokumenta ne smeju da se preklapaju. XML elementi mogu da sadrže atribute. Jedan element može da ima više jedinstvenih atributa. Atributi daju više informacije o XML elementu, oni definišu svojstva elementa. Atributi se zapisuju pored imena taga elementa u obliku para “naziv=vrednost”. Primer elementa sa atributom:

```
<naziv_taga naziv_atributa="vrednost_atributa"> ... </naziv_taga>
```

### Kod 2.3

Postoje definisana pravila za imenovanje elemenata i atributa:

- Imena su osetljiva na veličinu slova, naziv početnog i krajnjeg elementa mora biti isti.
- Za davanje imena mogu da se koriste: slova, cifre, donja crta \_, crtica -, dvotačka : i tačka .
- Ime može da počne slovom ili donjom crtom.
- Ne postoje rezervisane reči, sem reči “xml”.

XML dokument formira strukturu stabla, gde je svaki element čvor u stablu. XML document može da sadrži razne informacije. Primer XML dokumenta:



Slika 2.3: XML document [11]

**Document prolog (zaglavlje dokumenta)** dolazi iznad korenskog elementa. Ova sekcija sadrži XML deklaraciju koja sadrži detalje potrebne za XML procesor kako bi znao kako da parsira dokument.

**Document elements (elementi dokumenta)** su sastavni delovi XML dokumenta. Oni struktuiraju dokument u formu stabla.

## 2.5 WSDL

WSDL predstavlja standard za opisivanje web servisa. WSDL se formira kao XML dokument koji servis opisuje kao skup krajnjih pristupnih tačaka sa kojim se mogu razmenjivati poruke, koristeći sledeće elemente:

- **<types>** definiše tipove podataka koje koristi web servis
- **<message>** definiše poruke koje se razmenjuju za svaku operaciju
- **<portType>** definiše operacije koje mogu da se obave i poruke koje su uključene
- **<binding>** definiše protocol i format podataka za svaki <portType>

Na osnovu WSDL specifikacije je moguće generisati implementaciju servisa i klijentske klase za pristup servisu. WSDL se često koristi u kombinaciji sa SOAP-om i XML šemom da opiše web servise koji su dostupni putem interneta. Klijentski program koji se povezuje na web servis može da pročita WSDL da utvrdi koje funkcije su dostupne na serveru. Bilo koji specijalni tipovi podataka su opisani u WSDL fajlu u formi XML šeme. Klijent može da formira SOAP poruke da pozove jednu od funkcija koje su izlistane u WSDL fajlu. [12]

## 2.6 UDDI

UDDI specifikacija definiše registar za web servise. UDDI registar je web servis koji upravlja informacijama o provajderima, implementacijama i metapodacima web servisa. Provajderi servisa mogu da koriste UDDI kako bi registrovali svoje servise koje nude. Korisnik servisa može da pronađe servis u UDDI registru koji odgovara njegovim zahtevima i da dobije potrebne metapodake za korišćenje tog servisa. [14] Za pretragu i registrovanje web servisa koriste se SOAP poruke.

Kompanije mogu da registruju tri tipa informacija u UDDI registar. Ove informacije su sadržane u tri elementa UDDI registra, koji će biti objašnjeni u nastavku.

### 2.6.1 Bele stranice

Bele stranice sadrže osnovne informacije o kompaniji i njenom poslovanju. Sadrži osnovne informacije za kontakt kao što su: naziv, adresa, kontakt telefon i slično. Takođe, može da sadrži PIB (poreski identifikacioni broj). Ova informacija omogućuje pronalaženje web servisa koja nudi kompanija na osnovu jedinstvenog identifikatora kompanije.

### 2.6.2 Žute stranice

Žute stranice sadrže više informacija o kompaniji. Podrazumevaju informacije koje opisuju usluge web servisa pomoću svrstavanja u razne kategorije. Ovi podaci omogućavaju pronalaženje web servisa po vrsti proizvoda ili usluga koje kompanija nudi.

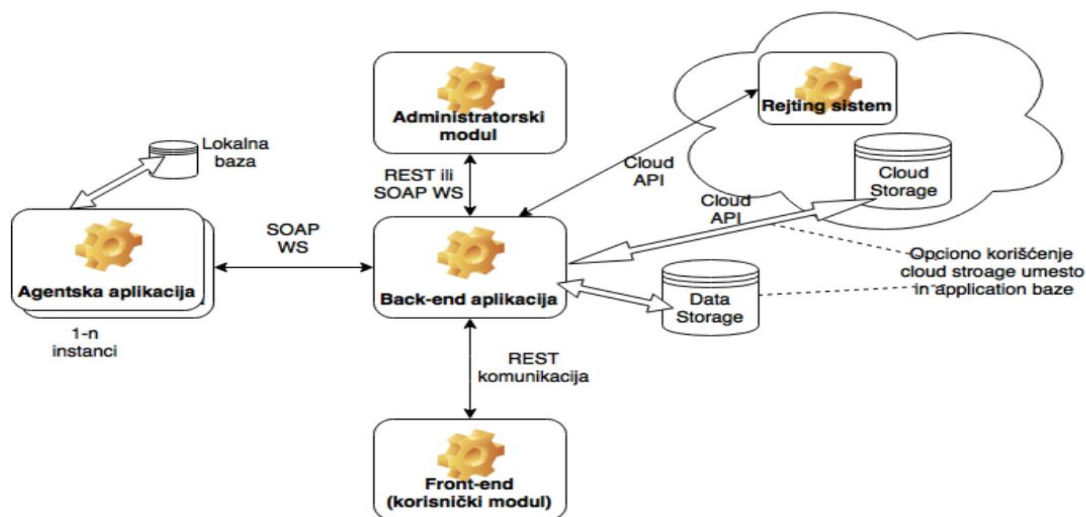
### 2.6.3 Zelene stranice

Zelene stranice omogućavaju povezivanje za uslugu web servisa nakon što je pronađen. Sadrže tehničke informacije koje opisuju funkcije web servisa, uključujući i podatke o lokaciji servisa. Ovi podaci omogućavaju pristup konkretnim web servisima date organizacije.

### 3. Opis arhitekture / model sistema

Realizovana web aplikacija zadužena je da omogući rezervisanje smeštaja putem interneta. Aplikacija je sastavljena od 4 modula:

1. Agentska aplikacija – web aplikacija koja je namenjena za agente smeštaja. Ima svoju lokalnu bazu koju sinhronizuje sa bazom glavne back-end aplikacije.
2. Back-end aplikacija – obavlja kompletnu poslovnu logiku i obezbeđuje servise ostalim modulima. Realizovana je kao servisno orijentisana web aplikacija.
3. Administratorski modul – nudi kontrolni panel za administratore sistema. Realizovana je kao front-end aplikacija putem Angular framework-a.
4. Korisnički modul – front-end aplikacija koja je realizovana putem Angular framework-a. Pruža interfejs za krajnjeg korisnika i sve funkcionalnosti koje su njemu potrebne.



Slika 3.0: Arhitektura sistema [17]

#### 3.1. Arhitektura sistema

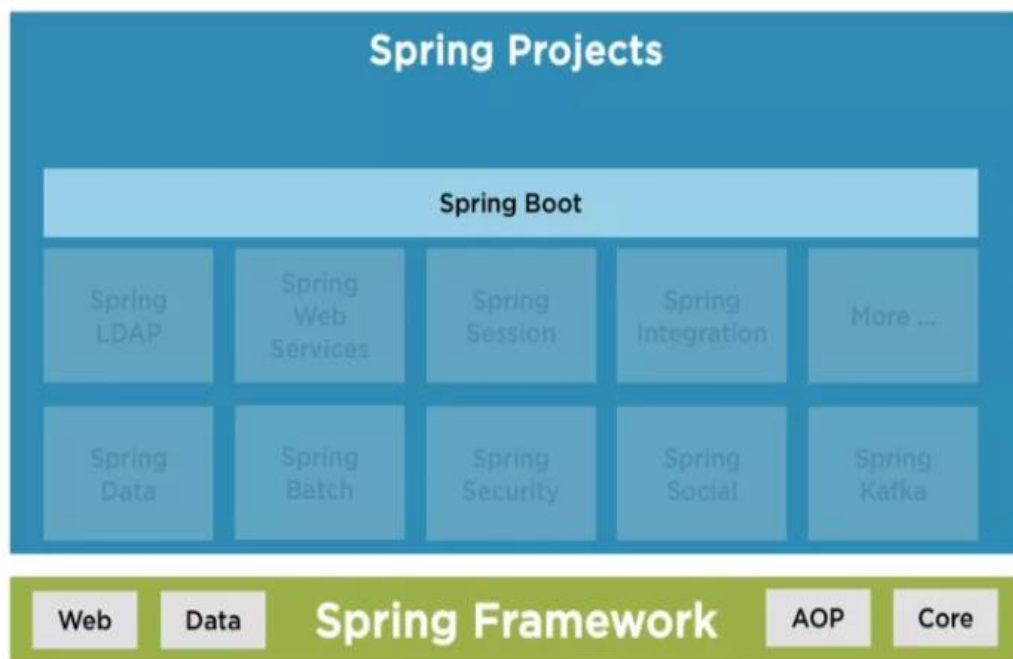
Glavna back-end aplikacija je realizovana putem **Spring Boot framework**-a koji predstavlja moćan alat za razvoj web aplikacija. Front-end aplikacija, odnosno korisnički modul, realizovan je putem **Angular framework**-a. **Angular** predstavlja jedan od najboljih alata za razvoj dinamičkih jednostraničnih (engl. single-page) aplikacija. *Front-end* i *back-*

end aplikacija komuniciraju putem **RESTful** web servisa, koji predstavlja težište ovog rada. Podaci sa kojima sistem radi se čuvaju lokalno na **MySQL** bazi podataka kao i na **Google Cloud** servisu koji služi za čuvanje rejtinga smeštaja.

### 3.1.1 Spring Boot

Spring Boot je najpopularniji okvir za razvoj aplikacija u Java ekosistemu koji je razvijen na vrhu **Spring framework-a** [21]. Razlog je zapravo prilično jednostavan – čini mnogo korisnih stvari i sa svakim novim izdanjem postaje sve bolji. Ne postoji mnogo tehnologija sa tako širokim dometom i stabilnošću. Danas Spring Boot predstavlja moderan i vrlo ekspresivan framework.

Spring Boot omogućava jednostavniji i brži način da se napravi, konfiguriše i pokrene web aplikacija. U **Spring Framework-u** je bilo potrebno da se sve stvari samostalno konfigurišu. To podrazumeva pravljenje mnogih XML dokumenata putem kojih se vrši konfiguracija. Ovo je jedan od glavnih problema koji Spring Boot rešava. Automatski konfiguriše sve funkcije koje želite da koristite i time pojednostavljuje proces razvoja aplikacije.

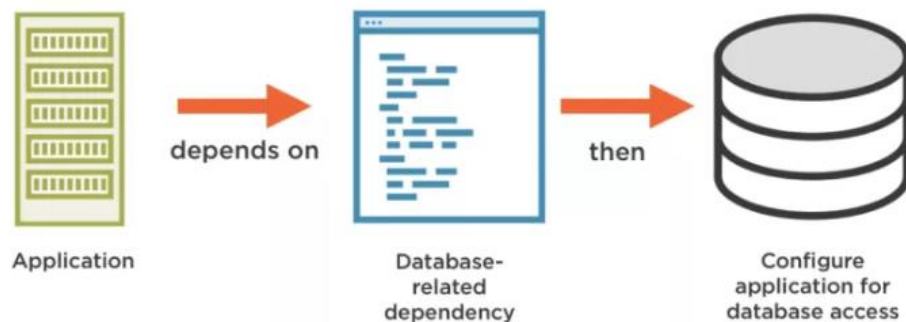


Slika 3.1: Spring Boot [22]

Najveće prednosti Spring Boot framework-a su:

- Auto-konfiguracija
- Samostalnost
- Generičke aplikacije

**Auto-konfiguracija** pokušava da samostalno konfiguriše aplikaciju na osnovu zavisnosti (engl. *dependency*) [23] koji se dodaju u projekat. Na primer, ako dodamo *dependency* koji se odnosi na bazu podataka, Spring Boot će pretpostaviti da verovatno želimo da koristimo bazu. Na osnovu toga, samostalno će konfigurisati aplikaciju da ima pristup bazi podataka.



**Slika 3.2:** Auto-konfiguracija [22]

Ako je *dependency* vezan za konkretnu bazu, može da napravi bolju predpostavku na osnovu konkretne baze i da odradi konfiguraciju koja je potrebna za nju.

Postavljanje **auto-konfiguracije** se vrši veoma lako uz pomoć **anotacije** [24]. Potrebno je dodati sledeće u Spring Boot aplikaciju:

```
@EnableAutoConfiguration ←
public class DemoApplication {
    ...
}
```

**Slika 3.3:** Primer omogućavanja auto-konfiguracije [22]

**Samostalnost** - proces pokretanja Java aplikacije zna da bude poprilično zamoran:

1. Pakovanje aplikacije
2. Biranje web servera koji želimo da koristimo
3. Konfigurisanje web servera

#### 4. Organizacija deployment-a i pokretanje web servera

Spring Boot u potpunosti olakšava ovaj proces. Potrebno je samo spakovati aplikaciju i pokrenuti je sa posebnom komandom. Ostatak Spring Boot sam odradi, pokreće i konfiguriše ugrađeni web server i preko njega aplikaciju.

**Generičke aplikacije** - Prilikom pravljenja Java web aplikacija postoje razne stvari koje se koriste prilikom razvoja (npr. biblioteke). U većini slučajeva programeri koriste najpopularnije biblioteke koje postoje za razvoj. Spring Boot omogućava rad sa **Spring Initializer**-om koji predstavlja web aplikaciju za kreiranje i konfigurisanje Spring baziranih aplikacija. **Spring Initializer** omogućava programerima da generišu inicijalnu strukturu projekta sa svim potrebnim bibliotekama.

#### 3.1.2 Angular framework

Angular je jedan od najpoznatijih JavaScript framework-a otvorenog koda za razvoj single-page web aplikacija. Ovaj framework je razvijen od strane Google kompanije. Angular koristi **Model-View-Controller (MVC)** arhitekturu, koja se koristi u razvoju web aplikacija.

- Model – Struktura podataka koja upravlja informacijama i prima ulaz od korisnika.
- View – Reprezentacija podataka. View je zapravo HTML.
- Controller – Odgovara na unos i vrši interakciju sa modelom.

Framework ima za cilj da odvoji aplikativnu logiku od **DOM (Document Object Model)** [25] manipulacije i da omogući dinamičko osvežavanje stranica. Uvodi mnogo moćnih funkcija koje omogućuju programeru da prilično lako stvara bogate single-page aplikacije.

Konkretno, uveden je zanimljiv koncept koji se zove **data binding** [26] koji predstavlja automatsko ažuriranje prikaza kad god se model (podaci) menja i obrnuto. Povrh toga, predstavljena je ideja **direktiva** [27] koje su omogućile formiranje sopstvenih HTML tagova koje je JavaScript oživeo. Još jedna bitna stvar koja je uvedena od strane Angulara je **dependency injection** [28], koja je omogućila spajanje komponenti aplikacije na način koji je olakšao kod za višekratnu upotrebu i testiranje.

Najveće prednosti Angular framework-a su sledeće:

- Angular ne pruža samo alate već i dizajn šablone za izgradnju projekta na izdrživ način. Ako se Angular aplikacija dobro razvije, ne završava se sa skupom klasa i metoda koje se teško mogu modifikovati, a još teže testirati. Kod je dobro struktuiran i nije potrebno trošiti vreme na to da se shvati šta se zapravo događa.



- Bolji je od JavaScript-a. Angular je razvijen pomoću **TypeScript-a** [31]. Nije potrebno u potpunosti učiti novi programski jezik, **TypeScript** predstavlja proširenje JavaScript-a.
- Nije potrebno samostalno izmišljati neke dodatne koncepte. Angular nudi mnogo alata za započinjanje izrade aplikacije odmah. Postoje direktive koje HTML elementima daju dinamično ponašanje. Moguće je poboljšati forme za unos podataka za raznim validacionim pravilima za polja za unos. Lako se mogu slati asinhroni pozivi raznih vrsta, kao i još mnogo dodatnih stvari koje framework nudi.
- Komponente su odvojene. Angular nastoji da ukloni čvrstu povezanost među komponentama aplikacije.
- **Data binding** koncept omogućava razdvajanje aplikativne logike i prezentacije podataka. Sve **DOM** manipulacije se dešavaju tamo gde treba.
- Angular je predviđen za temeljno testiranje. Podržava unit i end-to-end testiranje. [32]
- Angular je spreman za mobilne i desktop uređaje, što znači da je podržan od strane više platformi.

Može se reći da Angular ne predstavlja samo framework, već platformu koja programerima omogućava razvoj mobilnih, web i desktop aplikacija.

### 3.1.3 MySQL

MySQL je sistem za upravljanje bazama podataka sa otvorenim kodom koji se zasniva na struktuiranom jeziku upita (**Structured Query Language – SQL**). MySQL radi na skoro svim platformama: Linux, Windows, UNIX. Najčešće se vezuje za web aplikacije u kojima nalazi najveću primenu. Danas je MySQL iza mnogih vodećih web aplikacija, uključujući Facebook, Twitter i Youtube.

MySQL je zasnovan na klijent-server modelu. Njegovo jezgro je MySQL server, koji obrađuje sve instrukcije (ili naredbe). MySQL server je dostupan kao poseban program za upotrebu u klijent-server okruženju i kao biblioteka koja se može ugraditi (ili povezati) u posebne aplikacije.

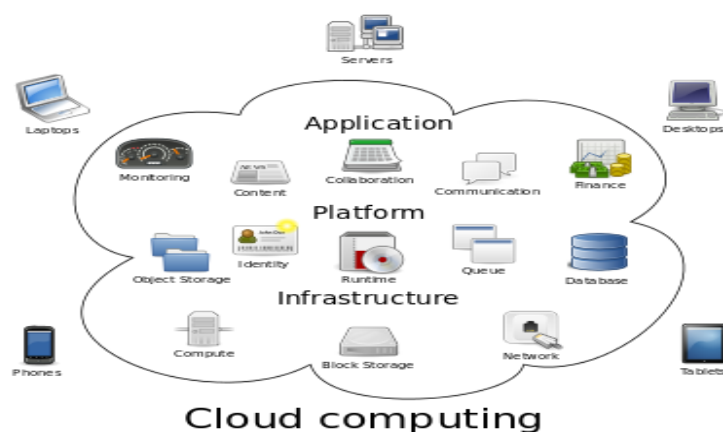
MySQL radi zajedno sa nekoliko programa koji podržavaju administraciju MySQL baze podataka. Komande se šalju MySQL serveru, koji ih obrađuje, putem MySQL klijenta, koji je instaliran na računaru.

Ovaj sistem za upravljanje bazama podataka je prvobitno razvijen da upravlja velikim bazama podataka. Iako je MySQL obično instaliran na jednoj mašini, on može da pošalje bazu podataka na više lokacija, jer korisnici imaju mogućnost da joj pristupe putem različitih

MySQL korisničkih interfejsa. Ovi interfejsi šalju **SQL** upite serveru na obradu, a zatim prikazuju rezultate. [34]

### 3.1.4 Google Cloud

**Cloud computing** je model isporuke računarskih resursa i skladišnih kapaciteta koji omogućava jednostavan, „na zahtev“ pristup konfigurabilnim računarskim resursima. Koncept **cloud computing-a** se oslanja na deljenje resursa preko mreže, najčešće interneta. Krajnji korisnici pristupaju resursima na **cloud-u** preko web pregledača ili desktop aplikacije na mobilnom telefonu. Prilikom pristupanja resursima, korisnik ne mora da zna tačnu lokaciju na kojoj se nalazi sistem koji pruža servis. Resursi se obezbeđuju u veoma kratkom vremenu, i čine dostupni korisniku bez komplikovanog upravljanja. **Cloud computing** obezbeđuje visok nivo apstrakcije procesnih resursa kao i resursa za skladištenje podataka.



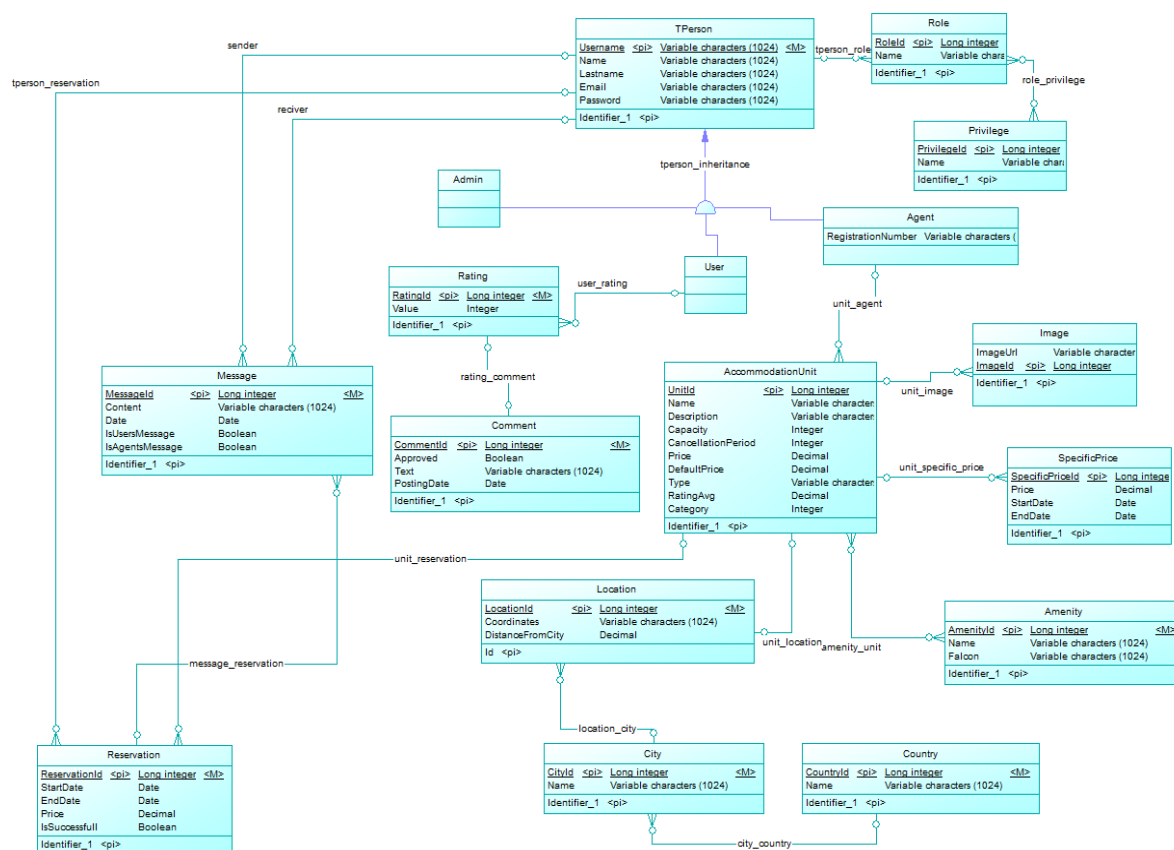
Slika 3.4: Cloud computing [35]

Google cloud platforma je skup javnih **cloud computing** servisa koje nudi Google. Platforma nudi razne servise za računanje, skladištenje i razvoj aplikacija koje se pokreću na Google hardveru. Uslugama Google Cloud Platform servisa mogu pristupiti programeri softvera, administratori **cloud-a** i drugi IT profesionalci putem javnog interneta ili neke namenske mrežne veze.

Za web aplikaciju, koja je namenjena rezervisanju smeštaja, je korišćen **Google Cloud Storage** servis. Ovaj servis je služio za skladištenje rejtinga koje korisnici ostave za smeštaj (ocena i komentari). Google Cloud Storage je dizajniran da čuva velike nestruktuirane skupove podataka. Takođe nudi opcije skladištenja baza podataka, **Cloud SQL** za MySQL bazu podataka.

## 3.2 Model sistema

Model sistema predstavljen je putem alata **PowerDesigner** [37] koji služi za modelovanje. Pravougaonici u dijagramu predstavljaju tabele (entitete) koji su deo sistema. U gornjem delu tabele nalazi se njen naziv, zatim slede atributi sa opisom tipa podatka (ukoliko je atribut u tabeli podvučen, to znači da on predstavlja identifikator).



Slika 3.5: Model podataka sistema

Na slici 3.5 prikazan je model sistema koji opisuje sve potrebne podatke koji su neophodni za izradu aplikacije. Model se može objasniti počevši od glavnog aktera u sistemu: korisnika. Svi podaci korisnika (kako i ostalih aktera u sistemu: Agent i Admin) se čuvaju u jednoj tabeli pod nazivom TPerson. Iskorišćen je mehanizam nasleđivanja **“jedna tabela po hijerarhiji nasleđivanja”**. Ovo znači da će se u bazi podataka podaci čuvati u jednoj tabeli (TPerson), koja uz osnovne kolone (podatke) za svakog aktera dobija jednu novu, dodatnu po imenu **diskriminatorksa kolona**. U **diskriminatorskoj koloni** čuva se informacija o tome koji je akter sistema u pitanju. Vrednosti **diskriminatorske kolone** mogu biti: User, Admin i Agent. Ovaj mehanizam je iskorišćen zbog ostalih aktera u sistemu, oni svi međusobno dele osnovne podatke. Pravljenje samog korisnika vrši se putem registracije. Osnovni podaci o korisniku su: korisničko ime, ime, prezime, e-mail adresa i šifra. Prvobitna uloga svakog korisnika, koja se daje prilikom registracije, je uloga krajnjeg korisnika sistema (User). Dodeljivanje uloga korisnika u sistemu implementirano je uz oslonac na **Spring Security** [38] framework. Uloga se svakom akteru dodeljuje putem vezivanja Role objekta sa određenim privilegijama (Privilege).

Moguće uloge korisnika sistema mogu biti:

- User – Krajnji korisnik može da pretražuje i rezerviše smeštaj, šalje poruke agentima i da ostavi komentar na smeštaj u kome je boravio.
- Agent – Vlasnici prostora primaju i upravljaju primljenim rezervacijama, postvaljaju nove smeštajne jedinice u ponudu, a mogu i samostalno uneti informacije da je određeni smeštaj zauzet tokom nekog perioda. Pored toga, Agenti pregledaju i odgovaraju na poruke klijentima koji su izvršili rezervaciju.
- Admin – Administratori sistema kao ključni zadatak imaju upravljanje poslovnim korisnicima/agentima, tj. da odobravaju agente smeštaja, kako bi ih registrovali u sistem i omogućili im da dodaju svoje smeštajne kapacitete u ponudu. Administrator sistema može i da blokira, aktivira i uklanja obične korisnike u sistemu. Takođe ima mogućnost za održavanje šifarnika dodatnih usluga smeštaja (wi-fi, klima, parking, TV, mini kuhinja, doručak, polu pansion, pansion, all inclusive, i sl.).

Za rezervaciju smeštaja bitan je korisnik koji vrši rezervaciju, smeštaj koji želi da rezerviše kao i početni i krajnji datum rezervacije. Model rezervacije predstavlja tabela Reservation, AccommodationUnit i TPerson sa međusobnim vezama. Nakon izvršene rezervacije korisniku se prikazuje lista svih napravljenih rezervacija, uključujući i najnoviju preko koje može da komunicira sa vlasnikom smeštaja. Poruke koje se razmene između korisnika i vlasnika smeštaja čuvaju se u tabeli Message. Za svaku poruku su bitni sledeći podaci: datum,

sadržaj poruke, indikator da li je poruku poslao korisnik ili vlasnik smeštaja, kao i veza prema rezervaciji kako bi znali na koju rezervaciju se odnosi prepiska. Pored komunikacije sa vlasnikom smeštaja putem poruka, korisniku se omogućava da ostavi rejting nakon boravka u smeštaju. Podaci o rejtingu se upisuju u tabelu Rating i podrazumevaju ocenu smeštaja i komentar. Podaci o komentaru se upisuju u posebnoj tabeli Comment koja se vezuju za tabelu Rating.

Podaci koji su potrebni za postavljanje nove smeštajne jedinice upisuju se u tabelu AccommodationUnit. Prilikom dodavanja nove smeštajne jedinice za nju se vezuje informacija o tome koji je agent izvršio dodavanje. Pored osnovnih podataka koje smeštajna jedinica ima, moguće je vezati dodatne usluge koje nudi i koji se nalaze u tabeli Amenity. Osnovni podaci poput lokacije i slika smeštajne jedinice se vezuju putem Location i Image tabela. Location tabela pored osnovnih podataka ima vezu sa City tabelom (preko koje se dobija informacija u kojoj državi se nalazi grad putem veze prema tabeli Country) kako bi se znalo o kom gradu i državi je reč. Takođe je moguće definisati terminski plan cena smeštajne jedinice. Podaci za terminski plan se upisuju u tabelu SpecificPrice i podrazumevaju cenu, datum početka i datum završetka.

Za funkcije administratora sistema potrebne tabele su Agent, User i Amenity. Prilikom registrovanja novog agenta podaci o njemu se upisuju u tabelu Agent. Za funkcije blokiranja, aktiviranja i uklanjanja korisnika potrebno je upisati informaciju o tome u obeležje status tabele User. Održavanje šifarnika se obavlja uz korišćenje tabele Amenity u koju se upisuju dodatne usluge koje smeštajna jedinica može da ponudi. Jedini podaci koji se upisuju jeste naziv dodatne usluge i link ka ikonici koja predstavlja grafički prikaz dodatne usluge.

Gore navedeni podaci, tabele i veze među tabelama je moguće videti na slici 3.5.

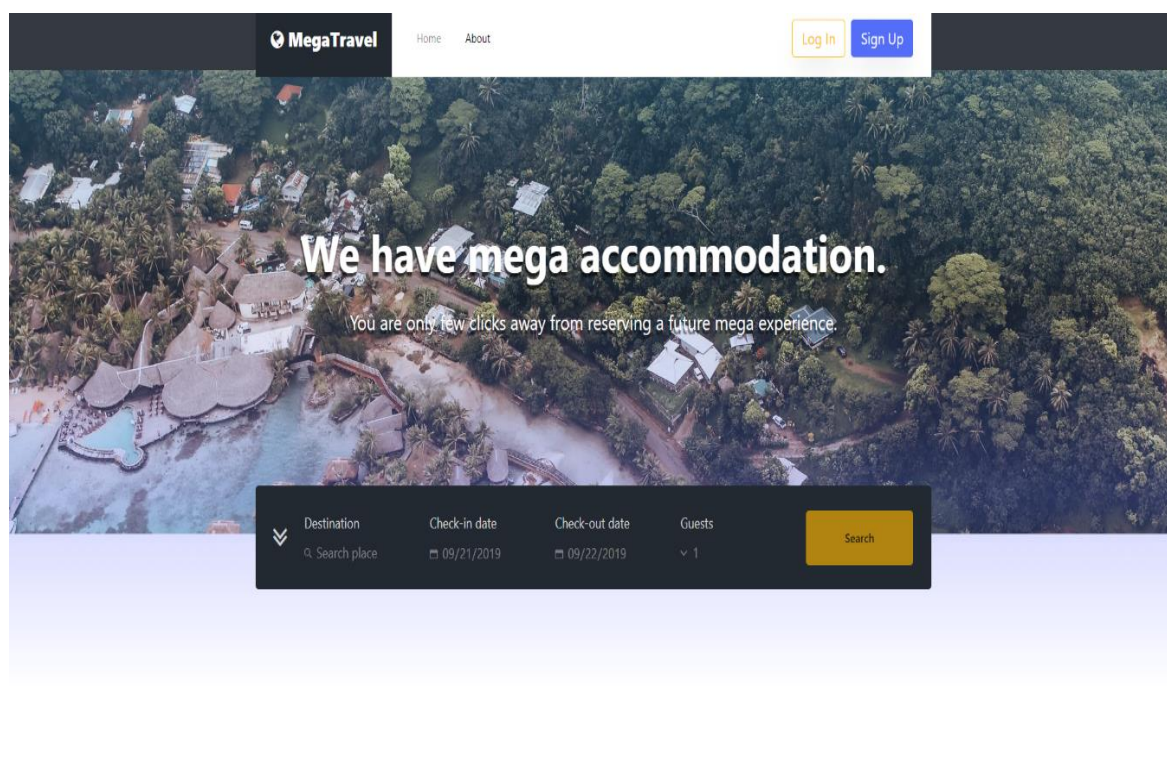


## 4. Implementacija


U ovom odeljku prikazani su i objašnjeni svi delovi implementacije koji su neophodni za komunikaciju putem REST-a između glavne back-end aplikacije i klijentske aplikacije. Detalji implementacije odnose se isključivo na postupku pretraživanja smeštaja i pravljenja same rezervacije. Sve ostale funkcionalnosti u sistemu implementirane su po uzoru na implementaciju koja će biti objašnjena u ovom odeljku.

### 4.2 Pretraga smeštaja

Mogućnost pretrage imaju i registrovani i neregistrovani korisnici sistema. Jedino registrovani korisnici mogu da prave rezervacije. Na slici 4.0 prikazana je osnovna stranica aplikacije na kojoj se nalazi forma za unos parametara pretrage.

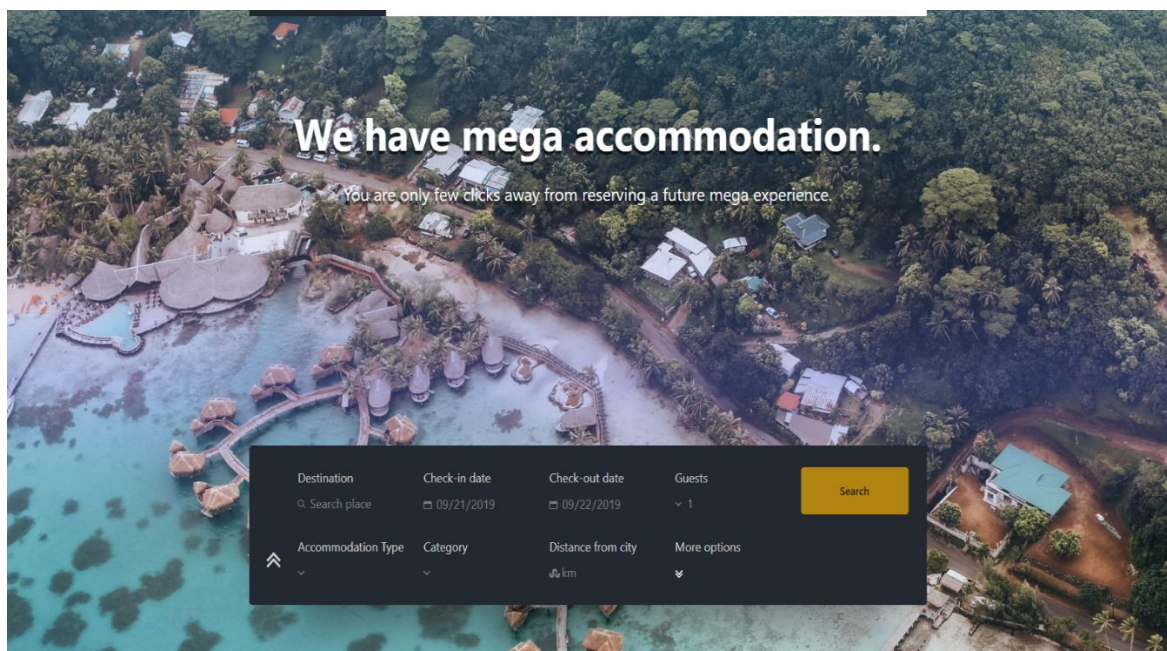


**Slika 4.0:** Osnovna stranica

Korisnik u osnovnoj pretrazi može da izabere sledeće parametre: destinaciju (Destination), datum početka (Check-in date), datum završetka (Check-out date) i broj gostiju (Guests). Pored toga, korisnik je u mogućnosti da izabere naprednu pretragu tako što klikne na .



Klikom na strelicu dobijaju se nove opcije za izbor parametara: tip smeštajne jedinice (Accommodation Type), kategorija (Category), daljina od grada (Distance from city), dodatne usluge koje smeštajna jedinica ima kao i broj dana za koji je moguće otkazati rezervaciju (More options). Primer takve forme vidi se na slici 4.1.



Slika 4.1: Napredna pretraga

Nakon što korisnik izabere parametre pretrage i klikne na dugme “Search”, šalje se zahtev sa klijenta serveru da vrati sve smeštajne jedinice koje odgovaraju parametrima.

Forma za pretragu je implementirana putem **Reactive** [40] forme koju nudi Angular. Parametri koje korisnik unese preuzimaju se i popunjavaju se vrednosti u poseban objekat koji će ih čuvati. Za prenos ovog objekta do servera koristi se **Data Transfer Object (DTO)** [41] dizajn šablon koji ide ruku uz ruku sa RESTful web servisima. Ovaj objekat prenosi se u JSON formatu putem mreže. Na slici 4.2 vidi se primer formiranja takvog objekta, uključujući i preuzimanje vrednosti parametara pretrage sa forme i dodeljivanje tih vrednosti objektu.



```
let searchDTO = {
  city: this.searchForm.value.destination,
  fromDate: this.searchForm.value.startDate,
  endDate: this.searchForm.value.endDate,
  personCount: this.searchForm.value.guests,
  cancellationPeriod: this.searchForm.value.cancellationPeriod,
  amenities: this.getSelectedAmenities(this.searchForm.get('amenities').value),
  type: this.searchForm.value.accommodationType,
  distanceFromCity: this.searchForm.value.distance,
  category : this.searchForm.value.category
}
```

**Slika 4.2:** DTO objekat

S obzirom da je korisniku omogućeno da izabere više dodatnih usluga od svih ponuđenih, postoji funkcija pod nazivom `getSelectedAmenities()` koja služi za preuzimanje svih obeleženih dodatnih usluga sa forme.

Nakon formiranja **DTO** objekta potrebno je poslati odgovarajući zahtev serveru. Za slanje zahteva u Angularu se prave posebne servisne klase koje su vezane za određen resurs. Ove servisne klase sadrže sve potrebne zahteve koje je moguće poslati serveru na obradu. Za ove potrebe izabrana je `POST` metoda i u telu zahteva se šalje objekat sa parametrima `searchDTO`. Primer funkcije koja je zadužena za slanje takvog zahteva vidi se na slici 4.3.

```
search(searchDTO) {
  return this.http.post("/api/accommodationsservice/accommodations/search", searchDTO);
}
```

**Slika 4.3:** Zahtev za pretragu smeštaja

URI je sastavljen na sledeći način:

- `/api` – ovaj deo putanje predstavlja **proxy** [42] putem kojeg smo definisali da sve zahteve šalje na onaj port na kom je pokrenuta naša aplikacija.
- `/accommodationsservice` – naziv aplikacije kojoj želimo da pošaljemo zahtev.
- `/accommodations` – naziv resursa.
- `/search` – naziv metode u kontroleru koja će obraditi zahtev.

Prilikom rada sa RESTful web servisima potrebno je ispoštovati predložene standarde za formiranje zahteva koji su opisani u poglavlju **0 REST**. U ovom slučaju se nije formirao zahtev prema preporučenim standardima. Razlog tome jeste što je moguće da korisnik unese veliki broj parametara pretrage i zato je korišćen **DTO** objekat za prenos svih parametara

putem POST metode, umesto da se koristi GET metoda i da se sve vrednosti parametara stavljaju u URI zahteva, što je preporučeno prema standardu.

Da bi se poslao zahtev potrebno je uvesti **HttpClient** [39] u Angular projekat. Nakon toga, da bi mogao da se koristi **HttpClient**, potrebno je izvršiti **dependency injection**.

```
constructor(private http: HttpClient, private router: Router) { }
```

**Slika 4.4:** Dependency injection

Ovo su stvari koje je bilo potrebno odraditi kako bi se sa klijenta mogao poslati zahtev po REST standardu. Potrebna implementacija koja je odrađena u glavnoj aplikaciji sledi u nastavku.

Kako bi bilo moguće prihvatiti zahteve koje klijent šalje, potrebno je registrovati kontroler koji će ih prihvatiti. U Spring Boot arhitekturi to se radi tako što se napravi nova klasa čije se ime daje tako što se napiše prvo naziv resursa sa kojim radimo i doda se reč „Controller“. U našem slučaju, koristi se `AccommodationUnitController` (`AccommodationUnit` predstavlja smeštajnu jedinicu, to je resurs sa kojim se radi) koji je potrebno pogoditi kako bi se vratile sve smeštajne jedinice koje odgovaraju parametrima pretrage. Da bi se ova klasa posmatrala kao kontroler potrebno je da se označi putem anotacije **@RestController**, koja se stavlja iznad definicije klase. Pored toga je potrebno definisati koji je naziv kontrolera pošto se ta informacija stavlja u URI zahteva. Ovo je postignuto putem anotacije **@RequestMapping**.

```
@RestController  
@RequestMapping("/accommodations")  
public class AccommodationUnitController
```

**Slika 4.5:** Rest kontroler

Rest kontroler sadrži sve funkcije koje je moguće izvršiti nad resursom smeštajne jedinice. Metoda koja izvršava pretragu smeštajne jedinice je formirana na sledeći način:

```
@RequestMapping(value = "/search", method = RequestMethod.POST)  
public ResponseEntity<?> search(@RequestBody ExtendedSearchDTO dto)
```

**Slika 4.6:** Metoda za pretragu

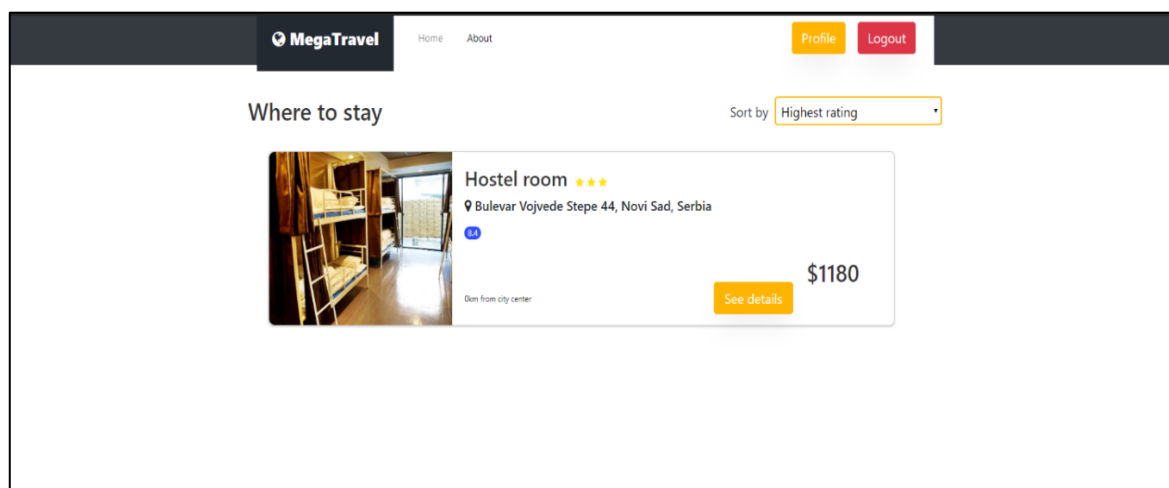
**@RequestMapping** anotacija ovde služi za mapiranje zahteva na metode kontrolera. U mapiranju, za **value** se stavlja naziv metode koji se gađa, dok se za **method** stavlja koji je tip HTTP metode. Sa anotacijom **@RequestBody** mapira se telo zahteva na odgovarajući objekat koji je poslat putem zahteva. Potrebno je da se taj objekat u potpunosti poklapa sa

onim koji je poslat (mora da ima iste attribute sa istim nazivima kao što je definisano u klijentu). Koristi se **DTO** objekat koji je objašnjen u gornjem delu, prilikom formiranja zahteva u klijentu.

## 4.2 Rezervacija smeštaja

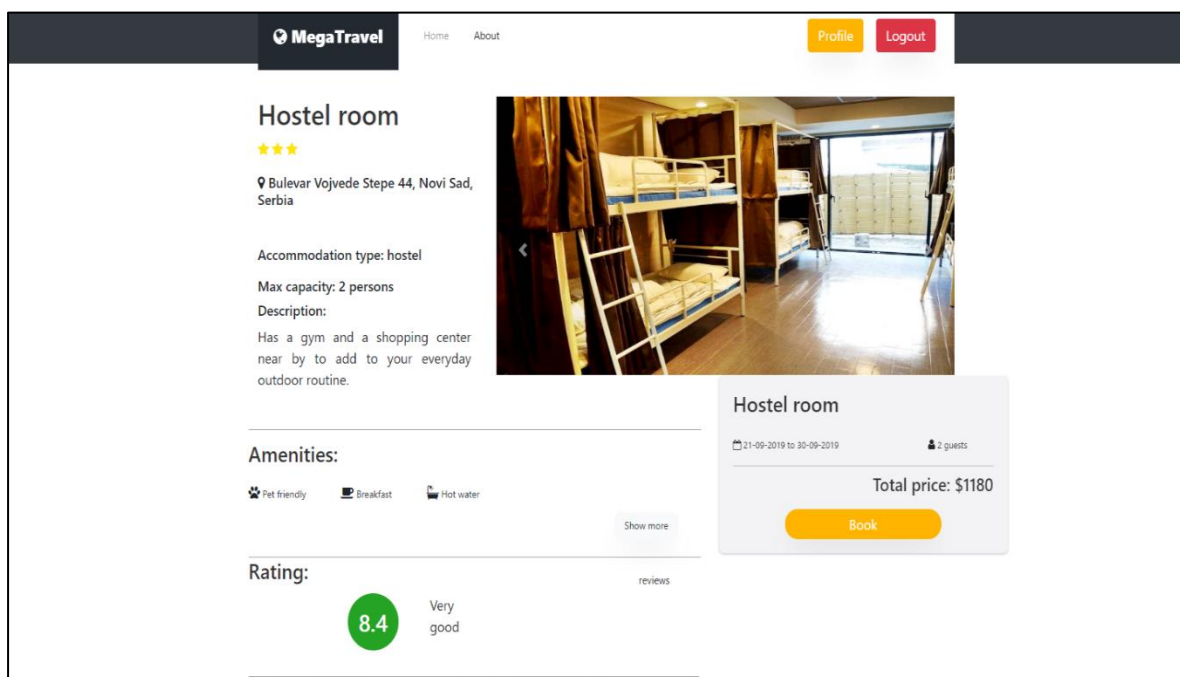
Za potrebe objašnjenja implementacije rezervacije smeštaja pretpostaviće se da je izvršena uspešna pretraga smeštaja sa sledećim parametrima pretrage: mesto Novi Sad, datum početka 20.09.2019., datum kraja 30.09.2019. i broj gostiju 2.

Nakon uspešnog slanja zahteva (čija je implementacija detaljno opisana u prethodnom poglavlju), korisniku se otvara nova stranica sa prikazom svih slobodnih smeštaja koji odgovaraju parametrima pretrage.



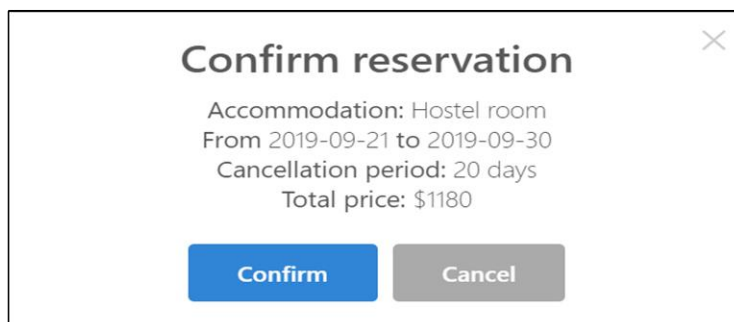
**Slika 4.7:** Rezultat pretrage smeštaja

Na slici 4.7 je prikazana lista smeštajnih jedinica koja odgovara rezultatu pretrage. Vidi se da je za date parametre pronađena jedna smeštajna jedinica. Smeštajna jedinica je prikazana kao kartica sa osnovnim informacijama. Dole u desnom uglu nalazi se dugme pod nazivom “See details” koja redirektuje korisnika na stranicu smeštajne jedinice.



**Slika 4.8:** Informacije smeštajne jedinice

Na slici 4.8 prikazana je stranica smeštajne jedinice. Na njoj se mogu videti osnovne informacije o smeštaju. Ispod slike smeštajne jedinice nalazi se mali prozor koji ima informacije o datumu za koji želi da se napravi rezervacija smeštaja, ukupnoj ceni koja treba da se plati i broj gostiju. Ako se pritisne dugme “Book” otvara se sledeći dijalog:



**Slika 4.9:** Dijalog rezervacije

Na slici 4.9 prikazan je dijalog za rezervaciju smeštaja. Na njemu se nalaze informacije o rezervaciji: koja je smeštajna jedinica u pitanju, od kada do kada se smeštaj rezerviše, period za koji može da se otkaže rezervacija (broj dana se gleda u odnosu na datum početka, što znači 20 dana pre početka same rezervacije) kao i ukupna cena. Ovaj dijalog implementiran je putem **sweetalert2** [44] biblioteke koja služi kao zamena za ugrađene JavaScript popup

dijaloge. Ova biblioteka iskorišćena je zbog svojih raznovrsnih funkcionalnosti, kao što je i mogućnost slanja zahteva i ubacivanja HTML koda za sopstveno kreiranje izgleda dijaloga.

```
Swal.fire({
  title: 'Confirm reservation',
  text: 'Do you wish to confirm your reservation?',
  html:
    '<b>Accommodation:</b> ' + this.acu.name +
    '<br><b>From</b> ' + this.sDate + ' <b>to</b> ' + this.eDate +
    '<br><b>Cancellation period:</b> ' + this.acu.cancellationPeriod + ' days' +
    '<br><b>Total price:</b> $' + this.totalPrice,
  showCloseButton: true,
  showCancelButton: true,
  focusConfirm: false,
  confirmButtonText:
    'Confirm',
  cancelButtonText:
    'Cancel',
```

**Slika 4.10:** Kod za definisanje izgleda dijaloga

Na slici 4.10 prikazana je implementacija izgleda dijaloga. Moguće je postaviti određene atribute dijaloga: naslov, tekst, HTML kod, prikaz dugmeta za zatvaranje, prikaz dugmeta za prekid akcije i tekst za navedena dugmeta.

Na slici 4.11 prikazana je implementacija slanja zahteva putem dijaloga, koji predstavlja nastavak koda sa slike 4.10:

```

    }).then((result) => {
      if (result.value) {
        let reservationDTO = {
          startDate: this.sDate,
          endDate: this.eDate,
          accommodationUnitId: this.id,
          price: this.totalPrice,
          reservator: this.user
        }

        this.resService.makeReservation(reservationDTO).subscribe(
          (data) => {
            Swal.fire({
              type: 'success',
              title: 'Your reservation was successful!',
              showConfirmButton: false,
              timer: 2000
            });
            this.router.navigate(['profile']);
          }, (error) => {
            if (error.status === 406) {
              Swal.fire({
                type: 'error',
                title: 'Unfortunately, someone has booked this accommodation in the meantime.',
                showConfirmButton: true
              });
              this.router.navigate(['home']);
            } else {
              Swal.fire({
                type: 'error',
                title: 'Something went wrong! Your reservation cannot be made.',
                showConfirmButton: true
              });
            }
          }
        );
      }
    });
  }
}

```

**Slika 4.11:** Slanje zahteva preko dijaloga

Ukoliko se pritisne dugme “Confirm” sa dijaloga inicira se slanje zahteva. Preuzimaju se potrebne informacije o rezervaciji (datum početka, datum kraja, identifikator smeštajne jedinice, ukupna cena, user koji je izvršio rezervaciju) i čuvaju se u **DTO** objektu reservationDTO. Nakon formiranja objekta, šalje se zahtev putem sledeće funkcije:

```

makeReservation(reservationDTO) {
  return this.http.post("/api/reservationservice/reservations", reservationDTO);
}

```

**Slika 4.12:** Zahtev za pravljenje rezervacije

S obzirom da pravimo novu rezervaciju, kreiramo novi resurs, potrebno je da izvršimo POST metodu i da pošaljemo novu rezervaciju putem tela zahteva. Potrebno je pogoditi kontroler pod imenom **reservations**, gde se nalazi metoda koja služi za kreiranje nove rezervacije.

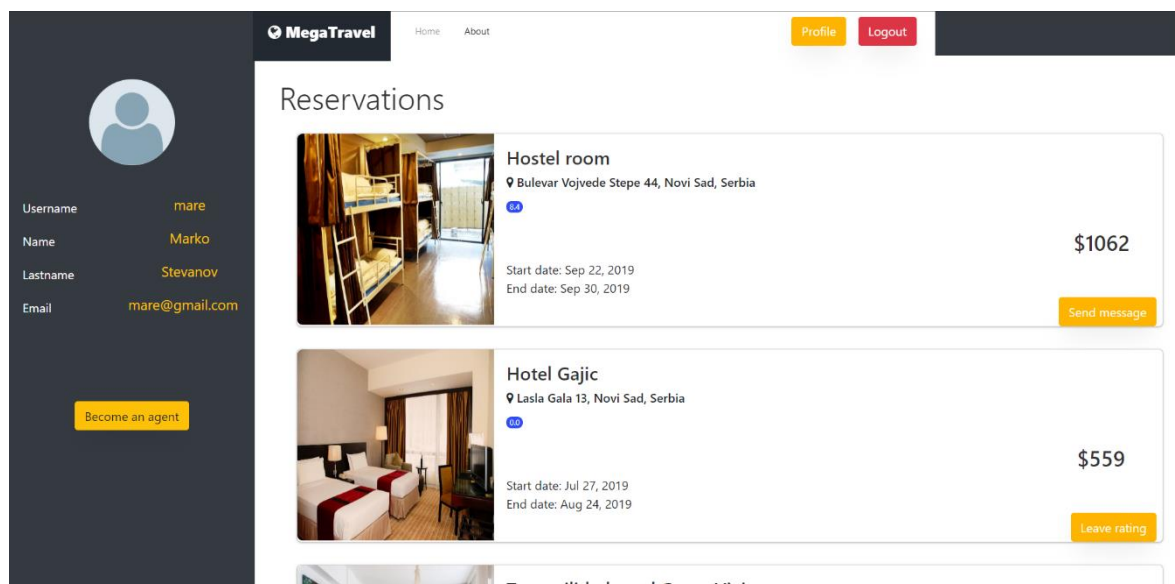
```

@RequestMapping(value = "", method = RequestMethod.POST)
public ResponseEntity<?> createReservation(@RequestBody ReservationDTO dto, HttpServletRequest request)

```

**Slika 4.13:** Metoda za pravljenje nove rezervacije

Nakon slanja zahteva i obrade zahteva, moguće je dobiti tri odgovora od strane servera. Ukoliko je rezervacija uspešno kreirana i sačuvana u bazu, korisnik se redirektuje na svoju profilnu stranicu i iskače mu dijalog koji ga informiše o uspešnosti kreiranja. Na profilnoj stranici se prikazuju osnovne informacije o korisniku kao i lista svih rezervacija, uključujući i najnoviju.



**Slika 4.14:** Lista svih rezervacija

Na slici 4.14 se može videti profilna stranica korisnika sa izlistanim rezervacijama. Za svaku rezervaciju se vide osnovne informacije o smeštajnoj jedinici, datumu početka i kraja i ukupnoj ceni. Pored toga postoji dugme “Send message” koje omogućuje korisniku da pošalje poruku agentu smeštajne jedinice. Ukoliko je već iskorištena rezervacija, umesto dugmeta “Send message” korisniku se prikazuje dugme “Leave rating” koje mu omogućava da ostavi ocenu i komentar za smeštajnu jedinicu.

U slučaju da nije bilo moguće napraviti rezervaciju, korisniku se putem iskaćućeg dijaloga prikazuje poruka o grešci i redirektuje se na početnu stranicu aplikacije.





## 5. Zaključak

U radu je opisan jedan od načina realizacije RESTful web servisa kod informacionog sistema koji je zadužen za rezervaciju smeštaja putem interneta.

Šira slika ovog sistema obuhvata sledeće funkcionalnosti: prijava korisnika na sistem, pretraga smeštaja, rezervacija smeštaja, ostavljanje komentara i ocene smeštaja nakon iskorišćene rezervacije, prepiska sa vlasnicima smeštaja, kreiranje nove smeštajne jedinice od stane vlasnika, odobravanje komentara korisnika od strane administratora kao i administracija agenata, korisnika i dodatnih usluga koje smeštajna jedinica poseduje. U ovom radu opisan je postupak pretrage smeštajne jedinice i pravljenje rezervacije.

Detalji implementacije rešavanog problema RESTful web servisa za komunikaciju glavne back-end i klijentske aplikacije dati su u poglavlju 4. u ovom radu. Detaljno je opisan postupak pretrage i rezervacije smeštaja, kao i sva potrebna implementacija, kako za klijentsku, tako i za glavnu back-end aplikaciju.

Za potrebe razvoja je iskorišćen Angular za korisničku aplikaciju, i Spring Boot okruženje za glavnu back-end aplikaciju. Ova okruženja predstavljaju moćne alate za razvoj web aplikacija i implementaciju RESTful web servisa.

Kroz opisanu implementaciju može se videti da se korišćenjem RESTful web servisa može postići jednostavan način komunikacije između aplikacija koje su u potpunosti nezavisne. Klijentska aplikacija treba samo da brine o tome da zahteve koje šalje dobro formira kako bi se na odgovarajući način poslali glavnoj aplikaciji na obradu, dok glavna aplikacija treba da brine samo o obradi zahteva i slanjem odgovarajućeg odgovara klijentu.

## 6. Literatura

- [1] What are web services?  
[https://www.tutorialspoint.com/webservices/what\\_are\\_web\\_services.htm](https://www.tutorialspoint.com/webservices/what_are_web_services.htm)
- [2] Slika 2.0: Struktura SOAP poruke  
<https://enastava.ftninformatika.com/courses/257/files/folder/predavanja?preview=109991>  
April 2019.
- [3] Web servisi i SOAP  
<https://enastava.ftninformatika.com/courses/257/files/folder/predavanja?preview=109991>  
April 2019.
- [4] Slika 2.1: Primer zahteva i odgovora <https://venkatmatta.com/2016/03/22/soap-webservices/> Mart 2016.
- [5] What are RESTful Web Services? <https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>
- [6] RESTful Web Services <https://www.guru99.com/restful-web-services.html>
- [7] Slika 2.2: Primer REST zahteva <https://dev.to/arpitmandliya/introduction-to-restful-web-services-22ga>
- [8] JSON – Introduction [https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)
- [9] XML tutorial <https://www.tutorialspoint.com/xml/index.htm>
- [10] Markup jezici i XML
- [11] Slika 2.3: XML dokument [https://www.tutorialspoint.com/xml/xml\\_documents.htm](https://www.tutorialspoint.com/xml/xml_documents.htm)
- [12] WSDL introduction [https://www.tutorialspoint.com/wsdl/wsdl\\_introduction.htm](https://www.tutorialspoint.com/wsdl/wsdl_introduction.htm)
- [13] XML WSDL [https://www.w3schools.com/xml/xml\\_wsdl.asp](https://www.w3schools.com/xml/xml_wsdl.asp)
- [14] What is UDDI? <http://uddi.xml.org/node/96>
- [15] UDDI – elements [https://www.tutorialspoint.com/uddi/uddi\\_elements.htm](https://www.tutorialspoint.com/uddi/uddi_elements.htm)
- [16] UDDI  
<https://enastava.ftninformatika.com/courses/257/files/folder/predavanja?preview=112667>
- [17] Slika 3.0: Arhitektura sistem  
<https://enastava.ftninformatika.com/courses/257/files/folder/projekat?preview=107128>
- [18] Introduction to Spring Framework <https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html> Avgust 2018. god.
- [19] Spring Framework Overview [https://www.tutorialspoint.com/spring/spring\\_overview.htm](https://www.tutorialspoint.com/spring/spring_overview.htm)  
Avgust 2018. god.
- [20] Spring Framework Architecture  
[https://www.tutorialspoint.com/spring/spring\\_architecture.htm](https://www.tutorialspoint.com/spring/spring_architecture.htm) Avgust 2018. god.

- [21] Spring Framework For Beginners <https://dzone.com/articles/spring-framework-tutorial-for-beginners-2>
- [22] What is Spring Boot <https://dzone.com/articles/what-is-spring-boot>
- [23] Introduction to Dependency Mechanism <https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html>
- [24] Java Annotations <https://www.javatpoint.com/java-annotation>
- [25] What is the Document Object Model? <https://www.w3.org/TR/REC-DOM-Level-1/introduction.html>
- [26] Data Binding <https://docs.angularjs.org/guide/databinding>
- [27] Directives <https://docs.angularjs.org/guide/directive>
- [28] Dependency injection <https://docs.angularjs.org/guide/di>
- [29] Angular introduction <https://www.sitepoint.com/angular-introduction/>
- [30] What is Angular? <https://www.hostinger.com/tutorials/what-is-angular>
- [31] TypeScript <https://en.wikipedia.org/wiki/TypeScript>
- [32] Testing <https://angular.io/guide/testing>
- [33] Introduction to services and dependency injection <https://angular.io/guide/architecture-services> Septembar 2018. god.
- [34] MySQL <https://searchoracle.techtarget.com/definition/MySQL>
- [35] Slika 3.4: Cloud computing [https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)
- [36] Google Cloud Platform <https://searchcloudcomputing.techtarget.com/definition/Google-Cloud-Platform>
- [37] PowerDesigner <https://en.wikipedia.org/wiki/PowerDesigner>
- [38] Spring Security <https://spring.io/projects/spring-security>
- [39] HttpClient <https://angular.io/guide/http>
- [40] Reactive forms <https://angular.io/guide/reactive-forms>
- [41] Data Transfer Object [https://en.wikipedia.org/wiki/Data\\_transfer\\_object](https://en.wikipedia.org/wiki/Data_transfer_object)
- [42] Proxy to Backend <https://github.com/angular/angular-cli/blob/master/docs/documentation/stories/proxy.md>
- [43] Spring RequestMapping <https://www.baeldung.com/spring-requestmapping>
- [44] Sweetalert2 <https://sweetalert2.github.io/>
- [45] HTTP methods <https://restfulapi.net/http-methods/>

## 7. Biografija

Kandidat Vladimir Cvetanović rođen je 29.09.1996. godine u Novom Kneževcu. Završio je srednju Ekonomsko-trgovinsku školu u Senti 2015. godine. Fakultet tehničkih nauka u Novom Sadu je upisao 2015. godine. Položio je sve ispite i ispunio sve obaveze koje su predviđene studijskim programom.