Capstone Project Report

Rent a place

BY

1. MANISH SHARMA
2. LALITHA SRI KAKARLA
3. RAJA BALA SIVA PRASAD
4. STUTI BANSAL
5. VARALAKSHMI CHENCHAMMAGARI

BATCH - JAVA_FSD_C4

Mentor: Mr. Shadab Ahmad Khan

ABSTRACT

Rent A Place is a Java Spring Boot, MySql and Angular Project which runs on the tomcat server, you can also run this project on Eclipse and Spring Tool Suite(STS). We have developed this Java Spring Boot and Angular Project on Rent A Place for automating the process of Rental a Place.

The Rent A Place is an E-Commerce Website which is used to Buy or Sell required places like Appartment ,villa, Hotel Rooms, Home etc. This project is developed to make booking of required place through online without direct interaction of the customer to the seller. The project is developed by using the technologies Angular, Spring boot and MySQL. The front is developed by using Angular, the backend is developed by using Spring boot and MySQL is the database which is used for data storing.

E-commerce brings convenience for customers as they do not have to leave home and only need to browse website online, especially for renting the property which are not sold in nearby shops.

TABLE OF CONTENT

INTRODUCTION

According to several reports, the rental sector is on a more robust path to substantial future development. Cities like Noida, Gurgaon, Mumbai, Bangalore, and Hyderabad are experiencing a strong rise in rental demand in Q1 (Jan-Mar) 2022 since they are home to a majority of corporate offices. Bangalore and Mumbai Metropolitan Region (MMR) were the most preferred cities for rentals over this time period, accounting for 27 percent and 30 percent of total demand, respectively. Rental housing demand in Hyderabad increased by 150 percent year on year and 30 percent quarter on quarter, respectively. Nonetheless, rates for mid-range and affordable flats have stayed stable in many cities, while luxury unit rentals have increased by 10-15 percent, mainly in places like Bangalore and Gurgaon.

At a time when more people are moving into apartments and residential buildings, the vitality of ondemand rental apps have gained popularity. People who cannot find a good property or those getting harassed by the brokers, should look forward to using the app. You will not have to pay any extra charges because the middlemen have been excluded from the scene.

Moreover, you also get a wide spectrum of choices to choose from when you are searching for a good property.

Technologies Used

Angular:

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your applications.

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components that are organized into NgModules. NgModules collect related code into functional sets; an Angular application is defined by a set of NgModules. An application always has at least a root module that enables bootstrapping, and typically has many more feature modules.

- Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data
- Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable and efficient.

Modules, components and services are classes that use decorators. These decorators mark their type and provide metadata that tells Angular how to use them.

- The metadata for a component class associates it with a *template* that defines a view. A template combines ordinary HTML with Angular *directives* and *binding markup* that allow Angular to modify the HTML before rendering it for display.
- The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).

An application's components typically define many views, arranged hierarchically. Angular provides the Router service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.

Springboot:

Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:

1. Autoconfiguration
2. An opinionated approach to configuration
3. The ability to create standalone applications

These features work together to provide you with a tool that allows you to set up a Spring-based application with minimal configuration and setup.

Spring is widely used for creating scalable applications. For web applications Spring provides
Spring MVC which is a widely used module of spring which is used to create scalable web applications.

But main disadvantage of spring projects is that configuration is really time-consume and can be a bit overwhelming for the new developers. Making the application production-ready takes some time if you are new to the spring.
Solution to this is Spring Boot. Spring Boot is built on the top of the spring and contains all the features of spring. And is becoming favourite of developer's these days because of it's a rapid production-ready environment which enables the developers to directly focus on the logic instead of struggling with the configuration and set up.
Spring Boot is a microservice-based framework and making a production-ready application in it takes very less time. Prerequisite for Spring Boot is the basic knowledge Spring framework.

MySQL:

MySQL is an open-source relational database management system (RDBMS). It is the most popular database system used with PHP. MySQL is developed, distributed, and supported by Oracle Corporation.

- The data in a MySQL database are stored in tables which consists of columns and rows.
- MySQL is a database system that runs on a server.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, and easy to use database system. It uses standard SQL
- MySQL compiles on a number of platforms.

MySQL follows the working of Client-Server Architecture. This model is designed for the end-users called clients to access the resources from a central computer known as a server using network services. Here, the clients make requests through a graphical user interface (GUI), and the server will give the desired output assoon as the instructions are matched. The process of MySQL environment is the same as the client-server model.

Problem Statement

RentAPlace is an online store to rent the home for short and long-time duration.
There are 2 users on the application: -
1. User
2. Owner

User Stories:

1. As a user I should be able to login, Logout and Register into the application.
2. As a user I should be able to see the top-rated properties in different categories.
3. As a user I should be able to search property as per the different search criteria's as: -
    - Check-in and check-out date
    - Place to visit
    - Type of property like (flat, apartment, villas)
    - Features like pool, pool facing, beach, beach facing, garden, garden facing and many more
4. As a user I should be able to see the selected properties in list or card view.
5. As a user I should be able to see 4 to 5 pic. Of each property.
6. As a user I should be able to reserve the property.
7. As a user I should be able to send message on the application to the owner.

Owner Stories –

1. As an Owner I should be able to login, Logout and Register into the application.
2. As an Owner I should be able to add properties on the platform. Properties can be more than one.
3. As an owner I should be able to update, delete and get the properties.
4. As an owner I should be able to see the messages of the users.
5. As an owner I should be able to reply on the messages.
6. As an owner I should be able to get mail as soon as the user reserved a property.
7. As an owner I should be able to conform to the reserved status.

Instructions –

1. Please use a folder on server to upload the images.
2. Please share the database structure in the .sql file.
3. Please create a separate microservice for sending and receiving messages.
4. Please use separate port to deploy the Angular UI and Spring Boot Microservice.
5. Please use the UI designing tool like (Bootstrap or Material) to make your UI better.
6. Please use Material UI to create the UI.

For Reference please visit:https://www.airbnb.co.in/
Sprint I Objectives

1. Create git repository
2. Create database schema (all tables along with their relationships)
3. Create entities in Spring
4. Create Microservice based structure.
5. CRUD on User and Owner
6. Create the Template in Angular(Static only) to hold images and property lists.

7. Develop Search Functionality in Angular
8. The property CRUD for owner

Sprint II Objectives

1. Create the owner Mail module and Owner confirm module.
2. Create Data Transfer objects
3. Create repository
4. Create Service layer logic
5. Create Controller to direct rest api
6. Create message Microservice

BackEnd Implementation:

1. Initializing Spring Boot:

To start with Spring Boot REST API, you first need to initialize the Spring Boot Project. You can easily initialize a new Spring Boot Project with Spring Initializer.

From your Web Browser, go to start.spring.io. Choose Maven as your Build Tool and Language as Java Select the specific version of Spring Boot you want to go ahead with.

Add dependencies:

1. Spring Web
2. Spring Boot DevTools
3. Spring Data JPA
4. MySQL Driver
5. JDBC API
6. Web Socket

2. Connecting Spring Boot to the Database:

Next, you need to set up the Database, and you can do it easily with Spring Data JPA.

Add some elementary information in your application.properties file to set up the connection to your preferred Database. Add your JDBC connection URL, provide a username and password for authentication, and set the ddl-auto property to update.

And also add server port within the application.properties by using server.port.

3. Create Required Packages:

Create new packages as per requirement with in the initiated project.

The packages are generated in the project are mentioned below:

1. Config
2. Controller
3. Exception
4. Model
5. Repository
6. Service

4. Develop the code as per requirement to the project:

The code respective to the backend mentioned within the Source Code Section.

FrontEnd Implementation:

Initially Install the Angular within the PC by using the command **npm install -g @angular/cli to globally install angular in your system**

After installation,

1. Creating Project:

Create an angular project by using the commad **ng new Project-Name**

The project is generated after the installation of required package. Open the project with the help of VisualStudio Code. After opening the project with the VS code develop the code as per the project requirement.

2. Creating Components:

For creating the components use the command **ng generate component component-name**.

The Components within the project are:

1. Add-question
2. addplace
3. admin-header
4. admindash
5. allusers
6. auth
7. chat
8. checkout
9. common
10. forbidden
11. home
12. locations
13. model

3. Execution:

The code for the components is mentioned within the Source Code section.

Make sure that app-routing.module.ts is well written and importing of respective component modules aredone perfectly.

After the completing of implementation part for executing the project by using the command **ng serve**

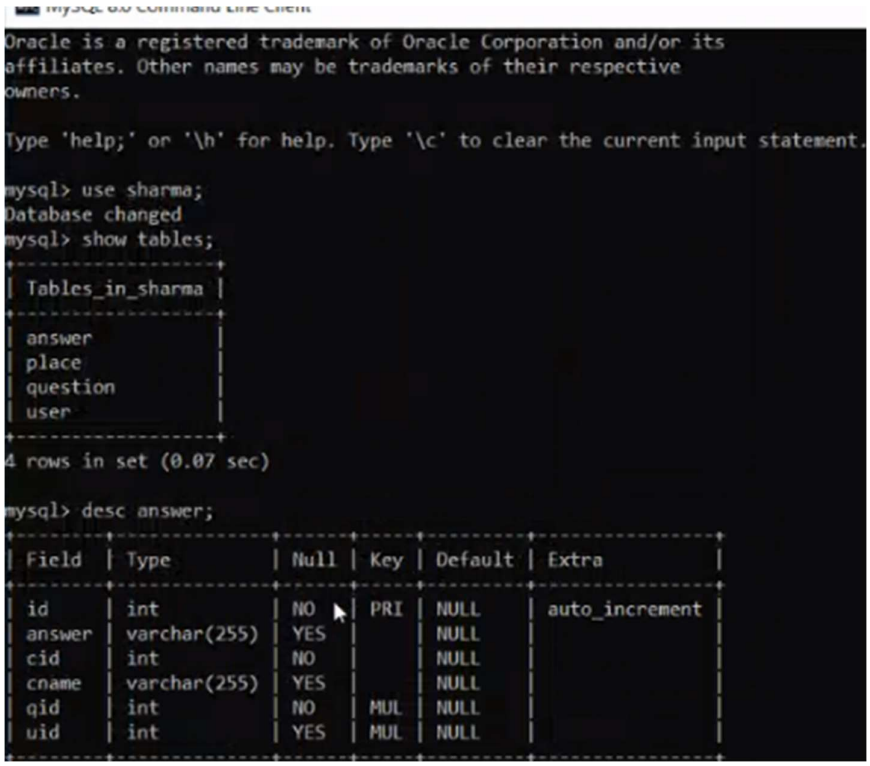
If the code is error free it executes perfectly by showing the port number.

Working with Database:

By the successful execution of backend developed project with the Spring Tool Suite then there is generation of data tables within the triggered database location.

After creation of data tables within the database location, Add admin details with in the admin tables by using the database query INSERT, like **insert into admin values('owner@gmail.com','1234').**

This is because of, the project don't have any admin registration so if we need to perform any operation with respect to admin we need to add admin details within the admin table in the database.



BackEnd:

BookingService:

Model:

Place:

```
package
com.bookingservice.model;

import javax.persistence.Entity;
import
javax.persistence.GeneratedValue;
import
javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Place {
	@Id
	@GeneratedValue(strategy =
GenerationType.IDENTITY)
	private int id;
	private String name;
	private float price;
	private String location;
	private String viewpoint;
	private String reserved;
	private String contact;
	private String Description;
	private String image1;
	private String image2;
	private String image3;
	private String image4;
	private int rating;
	private String type;
```

```
	public Place() {
		super();
		// TODO Auto-
generated constructor stub
	}

	public Place(String name,
float price, String location, String
viewpoint, String reserved, String
contact,
				String
description, String image1, String
image2, String image3, String
image4, int rating, String type) {
		super();
		this.name = name;
		this.price = price;
		this.location =
location;
		this.viewpoint =
viewpoint;
		this.reserved =
reserved;
		this.contact =
contact;
		Description =
description;
		this.image1 =
image1;
		this.image2 =
image2;
		this.image3 =
image3;
		this.image4 =
image4;
```

```
            this.rating = rating;
            this.type = type;
        }

        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        public String getName() {
            return name;
        }

        public void setName(String
name) {
            this.name = name;
        }

        public float getPrice() {
            return price;
        }

        public void setPrice(float
price) {
            this.price = price;
        }

        public String getLocation() {
            return location;
        }

        public void
```

```
setLocation(String location) {
                this.location =
location;
        }

        public String getViewpoint()
{
                return viewpoint;
        }

        public void
setViewpoint(String viewpoint) {
                this.viewpoint =
viewpoint;
        }

        public String getReserved()
{
                return reserved;
        }

        public void
setReserved(String reserved) {
                this.reserved =
reserved;
        }

        public String getContact() {
                return contact;
        }

        public void
setContact(String contact) {
                this.contact =
contact;
```

```
	}

	public String
getDescription() {
		return Description;
	}

	public void
setDescription(String description)
{
		Description =
description;
	}

	public String getImage1() {
		return image1;
	}

	public void setImage1(String
image1) {
		this.image1 =
image1;
	}

	public String getImage2() {
		return image2;
	}

	public void setImage2(String
image2) {
		this.image2 =
image2;
	}

	public String getImage3() {
```

```
		return image3;
	}

	public void setImage3(String
image3) {
		this.image3 =
image3;
	}

	public String getImage4() {
		return image4;
	}

	public void setImage4(String
image4) {
		this.image4 =
image4;
	}

	public int getRating() {
		return rating;
	}

	public void setRating(int
rating) {
		this.rating = rating;
	}

	public String getType() {
		return type;
	}

	public void setType(String
type) {
		this.type = type;
```

```
        }


}
```

Repository:

```
package
com.bookingservice.repository;

import java.util.List;

import
javax.transaction.Transactional;

import
org.springframework.data.jpa.re
pository.JpaRepository;
import
org.springframework.data.jpa.re
pository.Modifying;
import
org.springframework.data.jpa.re
pository.Query;
import
org.springframework.data.reposi
tory.query.Param;
import
org.springframework.stereotype.
Repository;

import
com.bookingservice.model.Place;

@Repository
public interface
bookingrepository extends
JpaRepository<Place, Integer> {

    @Modifying
```

```
        @Transactional
        @Query("update Place m
set m.contact =
:contact,m.price=:price where
m.id= :id")
        void
updateplace(@Param("contact")
String contact,@Param("price")
float price ,@Param("id") int
id);


        @Modifying
        @Transactional
        @Query("update Place m
set m.reserved =:status where
m.id = :id")
        void
updatestatus(@Param("id") int
id,@Param("status") String
status);


        @Query("select m from
Place m order by m.rating desc")
        List<Place>
findplacesBasedonrating();

        @Query("select m from
Place m order by m.rating asc")
        List<Place> getplacebt();

        @Modifying
        @Transactional
        @Query("update Place m
```

```
set m.name =:name where m.id =
:id")
        void
updateplacename(@Param("na
me") String
name,@Param("id") int id );

        @Modifying
        @Transactional
        @Query("update Place m
set m.price =:price where m.id =
:id")
        void
updateplaceprice(@Param("pric
e") float price,@Param("id") int
id);

        @Modifying
        @Transactional
        @Query("update Place m
set m.location =:loc where m.id =
:id")
        void
updateplacelocation(@Param("l
oc") String name,@Param("id")
int id);


        @Modifying
        @Transactional
        @Query("update Place m
set m.contact =:con where m.id =
:id")
        void
updateplacecontact(@Param("c
```

```
on") String name,@Param("id")
int id);


        @Modifying
        @Transactional
        @Query("update Place m
set m.Description =:desc where
m.id = :id")
        void
updateplacedesc(@Param("desc
") String name,@Param("id")
int id);


        @Modifying
        @Transactional
        @Query("update Place m
set m.rating =:rate where m.id =
:id")
        void
updateplacerating(@Param("rat
e") int name,@Param("id") int
id);

        @Modifying
        @Transactional
        @Query("update Place m
set m.type =:type where m.id =
:id")
        void
updateplacetype(@Param("type
") String name,@Param("id")
int id);
```

```
        @Query("select m from
Place m where m.location like
concat('%',:name,'%')")
        List<Place>
searchplace(@Param("name")
String name);

        @Query("select m from
Place m where m.viewpoint like
concat('%',:name,'%')")
        List<Place>
searchplacebyfeatures(@Param(
"name") String name);


}
```

Service:

```
package
com.bookingservice.service;

import java.util.List;

import
org.springframework.beans.fact
ory.annotation.Autowired;
import
org.springframework.stereotype.
Service;

import
com.bookingservice.model.Place;
import
com.bookingservice.repository.b
ookingrepository;

@Service
public class bookingservice {

	@Autowired
	private bookingrepository
bookingrepository;


//	method to add the place
	public Place
addplace(Place place) {
		return
bookingrepository.save(place);
	}

// method to update place
```

```
        public void
updateplace(String contact,float
price,int id) {

        bookingrepository.updatep
lace(contact,price,id);
        }

//method to reserve the place

        public void
reservePlace(int id,String status)
{

        bookingrepository.updates
tatus(id,status);
        }
// method to get the places based
on rating (top-bottom)

        public List<Place>
getsortedplacestb(){
                return
bookingrepository.findplacesBas
edonrating();
        }

//      method to get the places
based on rating (low-top)
        public List<Place>
getplacedbt(){
                return
bookingrepository.getplacebt();
        }
```

```
//      method to delete the place

        public void deleteplace(int
id) {
                Place p =
bookingrepository.getById(id);

        bookingrepository.delete(p
);
        }

//      method to get all the places
        public List<Place>
getallplaces(){
                return
bookingrepository.findAll();
        }

        public void
updateplacename(String name
,int pid) {

        bookingrepository.updatep
lacename(name, pid);
        }
        public void
updateplaceprice(float price ,int
pid) {

        bookingrepository.updatep
laceprice(price, pid);
        }

        public void
updateplacelocation(String name
```

```
,int pid) {

        bookingrepository.updatep
lacelocation(name, pid);
        }
        public void
updateplacecontact(String name
,int pid) {

        bookingrepository.updatep
lacecontact(name, pid);
        }
        public void
updateplacedesc(String name
,int pid) {

        bookingrepository.updatep
lacedesc(name, pid);
        }
        public void
updateplacerate(int rate ,int pid)
{

        bookingrepository.updatep
lacerating(rate, pid);
        }
        public void
updateplacetype(String name
,int pid) {

        bookingrepository.updatep
lacetype(name, pid);
        }

//      search place by location
```

```
	public List<Place>
searchplace(String place){
		return
bookingrepository.searchplace(p
lace);
	}

	public List<Place>
searchplacebyfeature(String
place){
		return
bookingrepository.searchplaceby
features(place);
	}

}
```

Controller:

```
package
com.bookingservice.controller;

import java.util.List;

import
org.springframework.beans.fact
ory.annotation.Autowired;
import
org.springframework.web.bind.a
nnotation.CrossOrigin;
import
org.springframework.web.bind.a
nnotation.DeleteMapping;
import
org.springframework.web.bind.a
nnotation.GetMapping;
import
org.springframework.web.bind.a
nnotation.PathVariable;
import
org.springframework.web.bind.a
nnotation.PostMapping;
import
org.springframework.web.bind.a
nnotation.PutMapping;
import
org.springframework.web.bind.a
nnotation.RequestBody;
import
org.springframework.web.bind.a
nnotation.RestController;
```

```
import
com.bookingservice.model.Place;
import
com.bookingservice.service.book
ingservice;

@RestController
@CrossOrigin
public class bookingcontroller {

	@Autowired
	private bookingservice
bookingservice;

	@PostMapping(value =
"addplace")
	public Place
addplace(@RequestBody Place
place) {
		return
bookingservice.addplace(place);
	}

	@PutMapping(value =
"updateplace/{contact}/{price}/{i
d}")
	public void
updateplace(@PathVariable("co
ntact") String Contact,
@PathVariable("price") float
price,

	@PathVariable("id") int
id) {
```

```
        bookingservice.updateplac
e(Contact, price, id);
        }

        @PutMapping(value="upd
atestatus/{id}/{status}")
        public void
updatestatus(@PathVariable("id
") int id,
@PathVariable("status") String
status) {

        bookingservice.reservePlac
e(id, status);
        }
        @DeleteMapping(value =
"deleteplace/{id}")
        public void
deleteplace(@PathVariable("id"
) int id) {

        bookingservice.deleteplace(
id);
        }


        @GetMapping("findallpla
ces")
        public List<Place>
getplaces(){
                return
bookingservice.getallplaces();
        }
        @PutMapping("updatepn
ame/{name}/{pid}")
```

```
	public void
updateplacename(@PathVariabl
e("name") String name
,@PathVariable("pid") int pid) {

	bookingservice.updateplac
ename(name, pid);
	}

	@PutMapping("updateppr
ice/{price}/{pid}")
	public void
updateplaceprice(@PathVariabl
e("price") float price
,@PathVariable("pid") int pid) {

	bookingservice.updateplac
eprice(price, pid);
	}

	@PutMapping("updateplo
c/{name}/{pid}")
	public void
updateplaceloc(@PathVariable(
"name") String name
,@PathVariable("pid") int pid) {

	bookingservice.updateplac
elocation(name, pid);
	}
	@PutMapping("updatepde
sc/{name}/{pid}")
	public void
updateplacedesc(@PathVariable
("name") String name
```

```
,@PathVariable("pid") int pid) {

	bookingservice.updateplac
edesc(name, pid);
	}
	@PutMapping("updatepra
te/{rate}/{pid}")
	public void
updateplacename(@PathVariabl
e("rate") int name
,@PathVariable("pid") int pid) {

	bookingservice.updateplac
erate(name, pid);
	}
	@PutMapping("updatepty
pe/{type}/{pid}")
	public void
updateplacetype(@PathVariable
("type") String name
,@PathVariable("pid") int pid) {

	bookingservice.updateplac
etype(name, pid);
	}
	@PutMapping("updatepco
ntact/{contact}/{pid}")
	public void
updateplacecontact(@PathVaria
ble("contact") String name
,@PathVariable("pid") int pid) {

	bookingservice.updateplac
econtact(name, pid);
	}
```

```
	@GetMapping("searchpla
ces/{place}")
	public List<Place>
searchplaces(@PathVariable("pl
ace") String place){
		return
bookingservice.searchplace(place
);
	}
	@GetMapping("searchpla
cesbyfeature"
				+ "/{place}")
	public List<Place>
searchplacesbyFeatures(@PathV
ariable("place") String place){
		return
bookingservice.searchplacebyfea
ture(place);
	}


	@GetMapping(value =
"top2bottom")
	public List<Place>
searchplacestopbottom(){
		return
bookingservice.getsortedplacestb
();
	}
	@GetMapping(value =
"bottom2top")
	public List<Place>
searchplacesbottomtop(){
		return
bookingservice.getplacedbt();
```

```
        }

    }
```

USER:

Answer:

```
package com.model;

import javax.persistence.Entity;
import
javax.persistence.GeneratedValu
e;
import
javax.persistence.GenerationTyp
e;
import javax.persistence.Id;

@Entity
public class Answer {
	@Id
	@GeneratedValue(strategy
= GenerationType.IDENTITY)
	private int id;
	private String answer;
	private int cid;
	private String cname;
	private int qid;
	public Answer() {
		super();
		// TODO Auto-
generated constructor stub
	}
	public Answer(String
answer, int cid, String cname,int
qid) {
		super();
		this.answer =
answer;
```

```
                this.cid = cid;
                this.cname =
cname;
                this.qid=qid;
        }
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getAnswer() {
                return answer;
        }
        public void
setAnswer(String answer) {
                this.answer =
answer;
        }
        public int getCid() {
                return cid;
        }
        public void setCid(int cid) {
                this.cid = cid;
        }
        public String getCname() {
                return cname;
        }
        public void
setCname(String cname) {
                this.cname =
cname;
        }
        public int getQid() {
                return qid;
```

```
        }
        public void setQid(int qid)
{
                this.qid = qid;
        }




}
```

Questions:

```
package com.model;

import java.util.ArrayList;
import java.util.List;

import
javax.persistence.CascadeType;
import javax.persistence.Entity;
import
javax.persistence.GeneratedValu
e;
import
javax.persistence.GenerationTyp
e;
import javax.persistence.Id;
import
javax.persistence.JoinColumn;
import
javax.persistence.OneToMany;

@Entity
public class Question {
	@Id
	@GeneratedValue(strategy
= GenerationType.IDENTITY)
	private int id;
	private String question;
	private int userid;
	private String username;
	@OneToMany(cascade =
CascadeType.ALL)
	@JoinColumn(name =
"qid")
```

```
	private List<Answer>
answers;
	public Question() {
		super();
		// TODO Auto-
generated constructor stub
	}

	public Question(String
question, int userid, String
username, List<Answer>
answers) {
		super();
		this.question =
question;
		this.userid = userid;
		this.username =
username;
		this.answers =
answers;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getQuestion()
{
		return question;
	}
```

```
	public void
setQuestion(String question) {
		this.question =
question;
	}

	public int getUserid() {
		return userid;
	}

	public void setUserid(int
userid) {
		this.userid = userid;
	}

	public String
getUsername() {
		return username;
	}

	public void
setUsername(String username) {
		this.username =
username;
	}

	public List<Answer>
getAnswers() {
		return answers;
	}

	public void
setAnswers(List<Answer>
answers) {
		this.answers =
```

```
answers;
        }
        public void
addanswer(Answer answer) {
                if(this.answers ==
null) {

        this.answers= new
ArrayList<Answer>();
                }

        this.answers.add(answer);
        }

}
```

Request:

```
package com.model;

public class request {
	private String username;
	private String password;


	public request(String username, String password) {
		super();
		this.username = username;
		this.password = password;
	}

	public request()
	{

	}

	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}


}
```

User.java

```
package com.model;

import java.util.ArrayList;
import java.util.List;

import
javax.persistence.CascadeType;
import javax.persistence.Entity;
import
javax.persistence.GeneratedValu
e;
import
javax.persistence.GenerationTyp
e;
import javax.persistence.Id;
import
javax.persistence.JoinColumn;
import
javax.persistence.OneToMany;

@Entity
public class User {
	@Id
	@GeneratedValue(strategy
= GenerationType.IDENTITY)
	private int id;
	private String username;
	private String password;
	private String phone;
	private String email;
	private String role;
	@OneToMany(cascade =
CascadeType.ALL)
```

```
	@JoinColumn(name =
"uid")
	private List<Question>
questions;
	@OneToMany(cascade =
CascadeType.ALL)
	@JoinColumn(name =
"uid")
	private List<Answer>
answers;
	public User() {
		super();
		// TODO Auto-
generated constructor stub
	}

	public User(String
username, String password,
String phone, String role,String
email) {
		super();
		this.username =
username;
		this.password =
password;
		this.phone = phone;
		this.role = role;
		this.email=email;
	}

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
```

```
        }
        public String
getUsername() {
                return username;
        }
        public void
setUsername(String username) {
                this.username =
username;
        }
        public String
getPassword() {
                return password;
        }
        public void
setPassword(String password) {
                this.password =
password;
        }
        public String getPhone() {
                return phone;
        }
        public void
setPhone(String phone) {
                this.phone = phone;
        }
        public String getRole() {
                return role;
        }
        public void setRole(String
role) {
                this.role = role;
        }

        public String getEmail() {
```

```
            return email;
        }

        public void setEmail(String
email) {
            this.email = email;
        }

        public List<Question>
getQuestions() {
            return questions;
        }

        public void
setQuestions(List<Question>
questions) {
            this.questions =
questions;
        }

        public List<Answer>
getAnswers() {
            return answers;
        }

        public void
setAnswers(List<Answer>
answers) {
            this.answers =
answers;
        }

        public void
addQuestion(Question question)
{
```

```
        if(this.questions==null) {

        this.questions= new
ArrayList<Question>();
                }

        this.questions.add(question
);
        }

        public void
addAnswer(Answer answer) {

        if(this.answers==null) {

        this.answers= new
ArrayList<Answer>();
                }

        this.answers.add(answer);
        }



}
```

User Repo:

```
package com.repo;

import java.util.List;

import
javax.transaction.Transactional;

import
org.springframework.data.jpa.re
pository.JpaRepository;
import
org.springframework.data.jpa.re
pository.Modifying;
import
org.springframework.data.jpa.re
pository.Query;
import
org.springframework.data.reposi
tory.query.Param;
import
org.springframework.stereotype.
Repository;

import com.model.Answer;
import com.model.Question;
import com.model.User;
import com.model.request;

@Repository
public interface userRepo
extends JpaRepository<User,
Integer> {

        User findByEmail(String
```

```
email);

        User
findByUsername(String
username);

        @Query("select m from
User m where m.username =
:username and m.password =
:password ")
        User
authenticateuser(@Param("user
name") String username,
@Param("password") String
password);

        void save(Question
question);

        @Query("select m from
Question m where m.id = :id")
        public Question
findbyid(@Param("id") int id);

        void save(Answer answer);

        @Query("select m from
Question m")
        List<Question>
getAllQuestion();

        @Query("select m from
Question m where m.userid =
:id")
        List<Question>
```

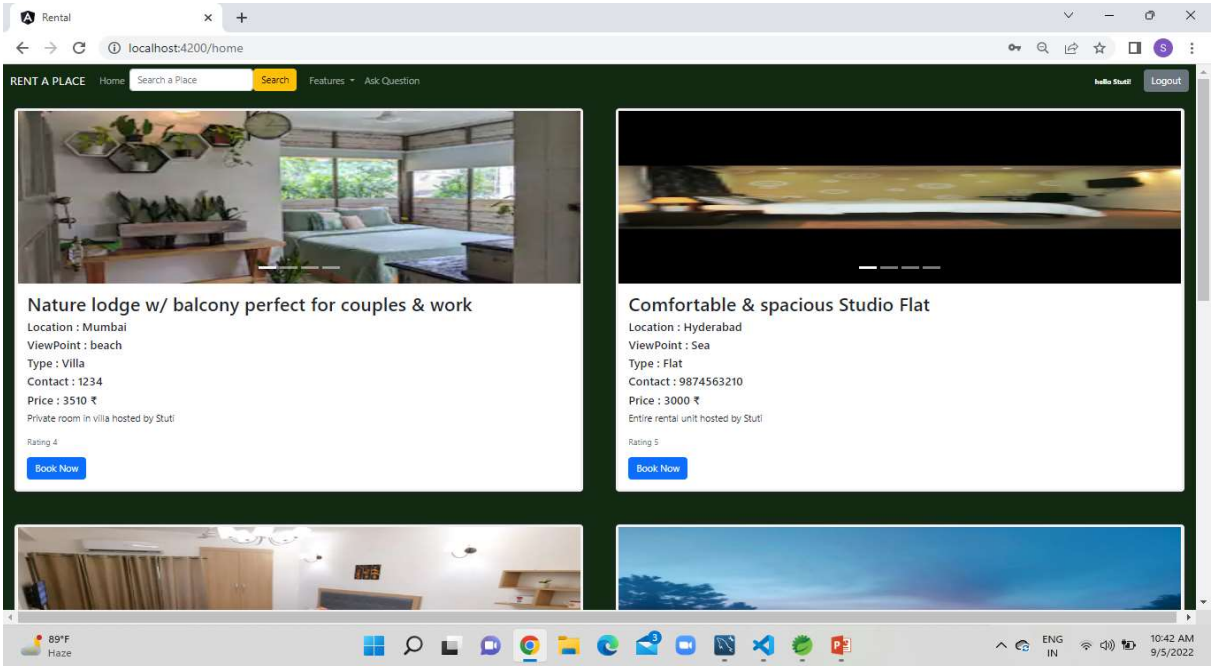
```
findUserQuestions(@Param("id
") int id);

        @Modifying
        @Transactional
        @Query("update User c
set c.phone =:phn where c.id =
:id")
        public void
updateUser(@Param("id") int
id,@Param("phn") String
phone);

        @Modifying
        @Transactional
        @Query("delete from User
c where c.id = :id")
        public void
deleteUser(@Param("id") int
id);

}
```

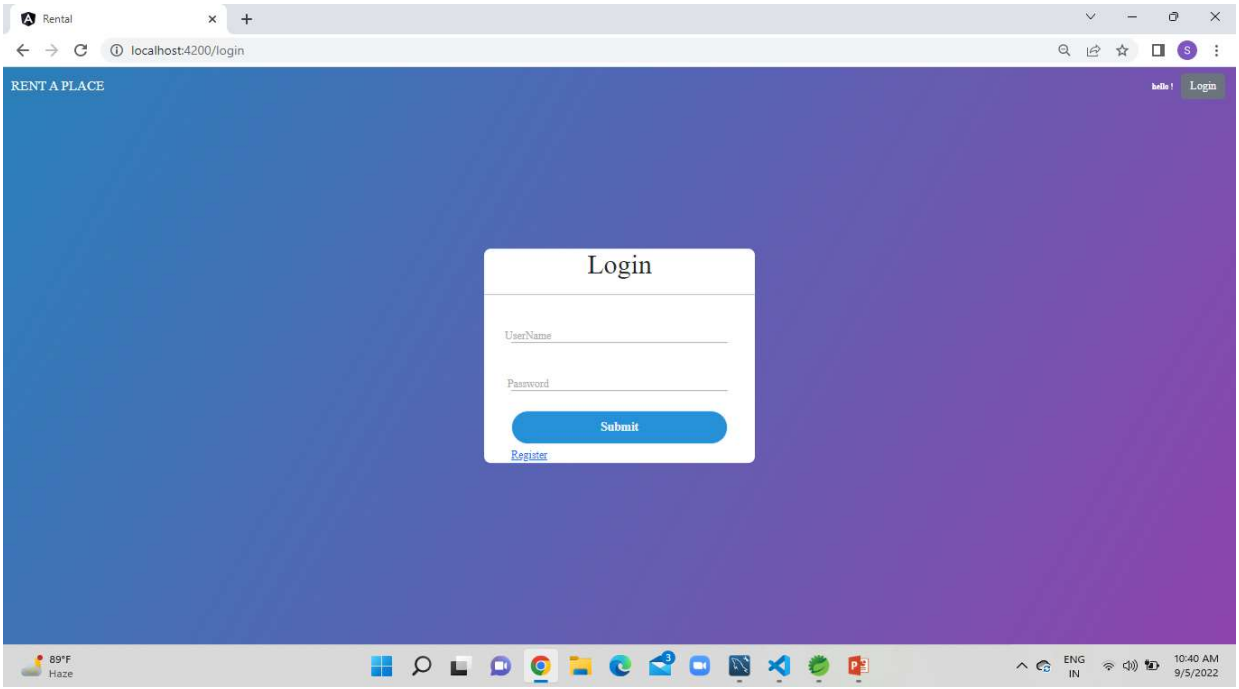
UserService:

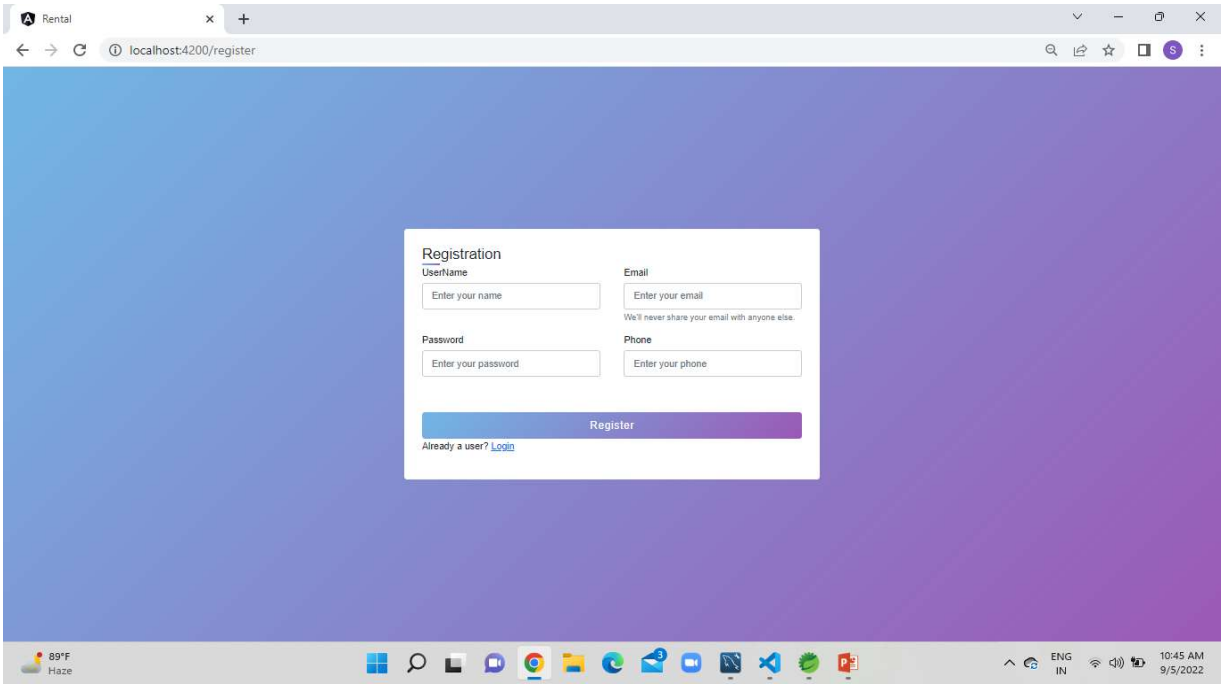
```
package com.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.model.Answer;
import com.model.Question;
import com.model.User;
import com.model.request;
import com.repo.userRepo;

@Service
public class Userservice {
	@Autowired
	public userRepo repo;

//	register the User
	public User registerUser(User user) throws Exception {
		User u1 = repo.findByEmail(user.getEmail());
		User u2 = repo.findByUsername(user.getUsername());
		if (u1 != null && u2 != null) {
			throw new Exception("User already Exist with " + user.getEmail());
		} else {
			return repo.save(user);
		}
	}

//	login to application
	public User loginUser(request request) throws Exception {
		User u = repo.authenticateuser(request.getUsername(), request.getPassword());
		if (u != null) {
			return u;
		} else {
			throw new Exception("incoorect password and email");
		}
	}

//	to get all the Users
	public List<User> getallUsers() {
		return repo.findAll();
	}

//	Adding Question
	public void addQuestion(Question question) throws Exception {
		User customer = repo.getById(question.getUserid());
		if (customer != null) {
			customer.addQuestion(question);
			repo.save(question);
		} else {
			throw new Exception("Customer is null");
		}
	}

//adding Answer
	public void addAnswer(Answer answer) throws Exception {
		User customer = repo.getById(answer.getCid());
		if (customer != null) {
			Question q = repo.findbyid(answer.getQid());
			if (q != null) {
```

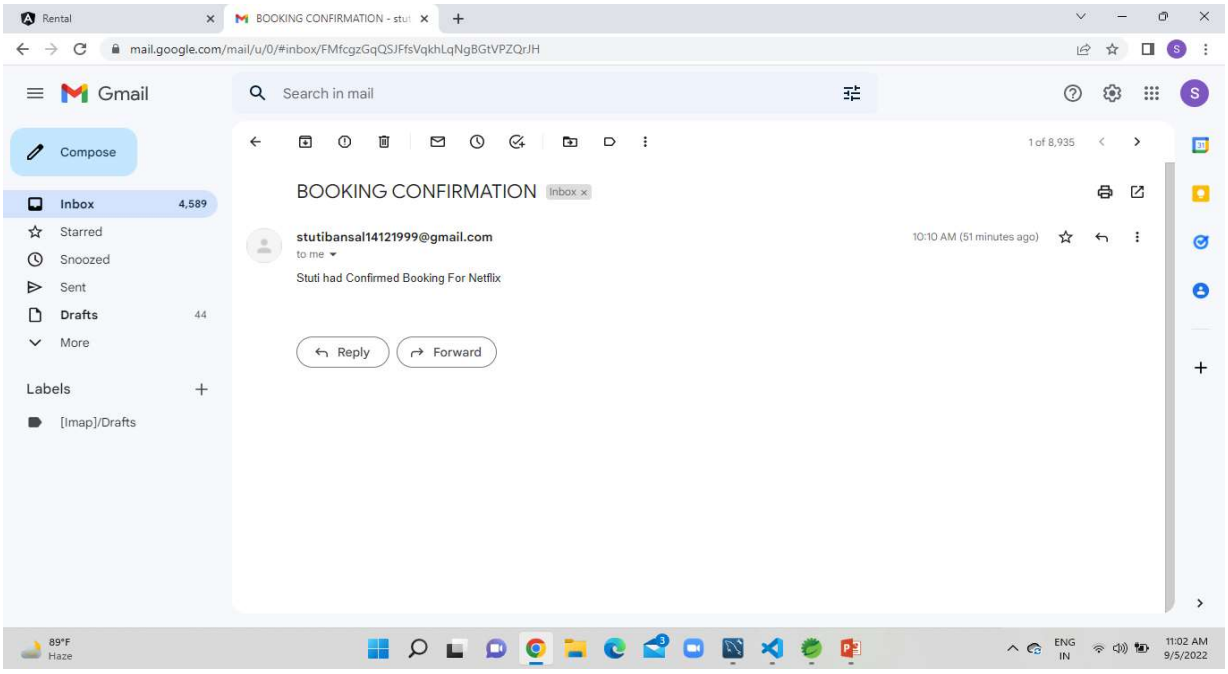
```
                        customer.addAnswer(answer);
                        q.addanswer(answer);
                        repo.save(answer);
                  } else {
                        throw new Exception(" Question is null");
                  }

            } else {
                  throw new Exception("Customer is null");
            }
      }

//to get all Question on console

      public List<Question> getallQuestions() {
            return repo.getAllQuestion();
      }

//    to get User questions
      public List<Question> getUserQuestions(int id){
            return repo.findUserQuestions(id);
      }

      //to update user
      public void updateUser(int uid,String phone) {
            repo.updateUser(uid, phone);
      }
      //to delete customer
      public void deleteCustomer(int id) {
            repo.deleteUser(id);
      }

}
```

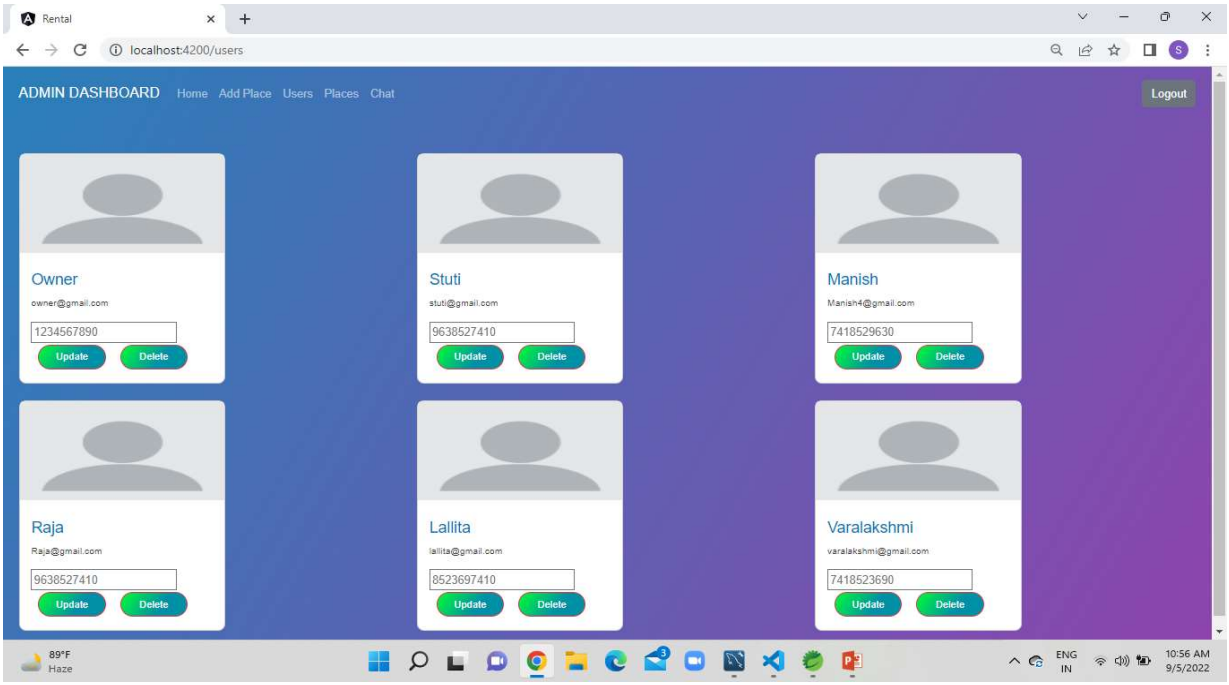
Output:

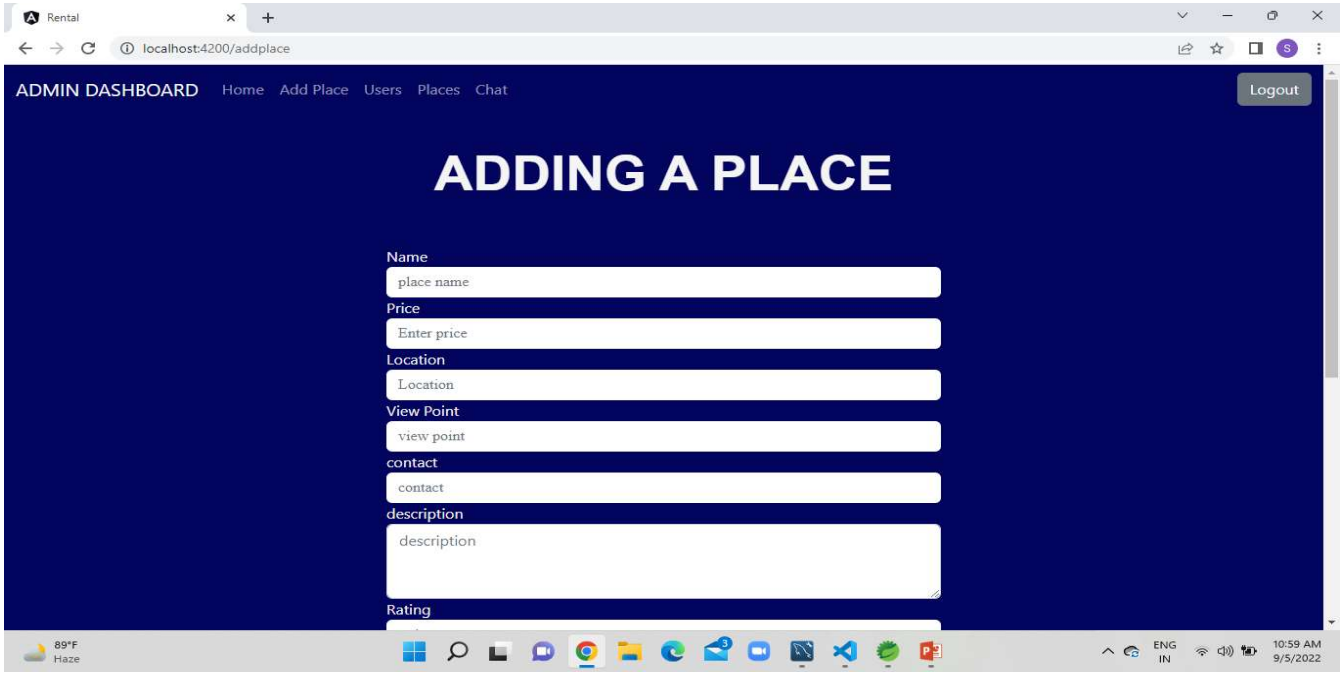

Rental
localhost:4200/home
RENT A PLACE
Home
Search a Place
Search
Features
Ask Question
hello Stuti!
Logout
Nature lodge w/ balcony perfect for couples & work
Location : Mumbai
ViewPoint : beach
Type : Villa
Contact : 1234
Price : 3510 ₹
Private room in villa hosted by Stuti
Rating 4
Book Now
Comfortable & spacious Studio Flat
Location : Hyderabad
ViewPoint : Sea
Type : Flat
Contact : 9874563210
Price : 3000 ₹
Entire rental unit hosted by Stuti
Rating 5
Book Now
89°F
Haze
ENG
IN
10:42 AM
9/5/2022


Rental
localhost:4200/login
RENT A PLACE
hello !
Login
Login
UserName
Password
Submit
Register
89°F
Haze
ENG
IN
10:40 AM
9/5/2022


Rental
localhost:4200/register
Registration
UserName
Enter your name
Email
Enter your email
We'll never share your email with anyone else.
Password
Enter your password
Phone
Enter your phone
Register
Already a user? Login
89°F
Haze
ENG
IN
10:45 AM
9/5/2022


Rental
BOOKING CONFIRMATION - stut
mail.google.com/mail/u/0/#inbox/FMfcgzGqQSJFfsVqkhLqNgBGtVPZQrJH
Gmail
Search in mail
Compose
Inbox 4,589
Starred
Snoozed
Sent
Drafts 44
More
Labels
[Imap]/Drafts
1 of 8,935
BOOKING CONFIRMATION
Inbox
stutibansal14121999@gmail.com
to me
10:10 AM (51 minutes ago)
Stuti had Confirmed Booking For Netflix
Reply
Forward
89°F
Haze
ENG
IN
11:02 AM
9/5/2022


Rental
localhost:4200/users
ADMIN DASHBOARD
Home
Add Place
Users
Places
Chat
Logout
Owner
owner@gmail.com
1234567890
Update
Delete
Stuti
stuti@gmail.com
9638527410
Update
Delete
Manish
Manish4@gmail.com
7418529630
Update
Delete
Raja
Raja@gmail.com
9638527410
Update
Delete
Lallita
lallita@gmail.com
8523697410
Update
Delete
Varalakshmi
varalakshmi@gmail.com
7418523690
Update
Delete
89°F
Haze
ENG
IN
10:56 AM
9/5/2022


Rental
localhost:4200/addplace
ADMIN DASHBOARD
Home
Add Place
Users
Places
Chat
Logout
ADDING A PLACE
Name
place name
Price
Enter price
Location
Location
View Point
view point
contact
contact
description
description
Rating
89°F
Haze
ENG
IN
10:59 AM
9/5/2022