A PROJECT REPORT ON

TOURISM MANAGEMENT SYSTEM

SUBMITTED BY

Ms. Nivetha S

Ms. Vishnupriya D

Ms. Pavithra G

Ms. Poonam Muttagikar

Ms. Monika Parasuraman

Batch No: 6769 and 6773

Under the Guidence of

Trainer Mrs. Indrakka Mali

INDEX

1. Introduction

The project “Tourist Management System” is developed using spring boot framework, which mainly focuses on basic operations of visitor information system. Like Inserting, Deleting, Updating and getting all records of visitors introduction.

Admin Module:

- In the Admin Module:
- We can enter the Admin Name and Password for Admin.
- We can give visitor personal details in the Admin module.
- We can get the visitors details and delete by record by using visitors Id and Name.

Visitors Module:

- In the visitor module the visitors can perform:
- Fetch all visitor’s records.
- Fetch visitor record by visitor Id.
- Fetch visitor record by visitor EmailId.
- Fetch visitor record by visitor Name.
- Fetch visitor record by Address and Mobile Number.
- One visitor can eligible to apply for many tourist places.

2. Objectives:

- It provides “better and efficient” service.
- Faster way to get information about the visitors.
- Provide facility for proper Accommodation and Package details.
- All details will be available on a click.

System Overview:

- The Tourism management system will be automated the traditional system.
- There is no need to use paper and pen.
- Checking visitor details is easy task here.
- Adding new visitor record and easy to get the records.
- Deleting and Updating a record of a particular visitor easily.

3. Requirement

Software Requirement:

- Database : MySQL
- API - Spring Data JPA, spring web, spring security
- Tools : Postman, IDE-Spring Tool Suit4
- language : Java 8
- Operating System : Windows 10

Hardware Requirement:

- RAM : 8GB
- Processor : Intel Core i3
- Memory : 700MB
- Disk Space : 1TB

4. System overview and snapshots

Spring Tool Suit:

- STS is an Eclipse-based development environment that is customized for the development of spring applications.
- It provides a ready-to-use environment to implement, debug, run and deploy your applications.

Postman:

Postman is a standalone software testing API (Application Programming Interface) platform to build, test, design, modify, and document APIs. It is a simple Graphic User Interface for sending and viewing HTTP requests and responses.

MySQL:

- MySQL is a relational database management system.
- MySQL is open-source software.
- MySQL is free to use.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, scalable, and easy to use.
- MySQL is cross-platform.

Annotations:

1.@Service:

We mark beans with @Service to indicate that they're holding the business logic. Besides being used in the service layer, there isn't any other special use for this annotation.

2. @Repository:

@Repository's job is to catch persistence-specific exceptions and re-throw them as one of spring's unified unchecked exceptions.

3. @Autowired:

The spring framework enables automatic dependency injection. In other words, by declaring all the bean dependencies in a spring configuration file, Spring container can auto wired relationships between collaborating beans. This is calledspring bean auto wired.

4. @GetMapping:

The @GetMapping annotation is a specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping (method = RequestMethod.GET).

5. @PostMapping:

The @PostMapping is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST). The @PostMapping annotated methods in the @Controller annotated classes handle the HTTP POST requests matched with given URI expression.

6. @PutMapping:

The @PutMapping annotation in Spring maps the HTTP PUT requests onto specific handler methods. It is a shortcut for @RequestMapping(method = RequestMethod. PUT). The @RequestMapping annotation is used for mapping all the incoming HTTP request URLs to the corresponding controller methods.

7. @DeleteMapping:

The @DeleteMapping annotation maps HTTP DELETE requests onto specific handler methods. It is a composed annotation that acts as a shortcut for @RequestMapping(method= RequestMethod. DELETE).

8. @OneToMany:

A one-to-many relationship between two entities is defined by using the @OneToMany annotation in Spring Data JPA. It declares the mappedBy element to indicate the entity that owns the bidirectional relationship. Usually, the child entity is one that owns the relationship and the parent entity contains the @OneToMany annotation.

9. @Generated Value:

Marking a field with the @GeneratedValue annotation specifies that a value will be automatically generated for that field. This is primarily intended for primary key fields but Object DB also supports this annotation for non-key numeric persistent fields as well.

10. @PathVariable:

The @PathVariable annotation can be used to handle template variables in the request URI mapping, and set them as method parameters.

11. @Override:

@Override annotation informs the compiler that the element is meant to override an element declared in a super class. Overriding methods will be discussed in Interfaces and Inheritance. While it is not required to use this annotation when overriding a method, it helps to prevent errors.

12. @Entity:

The @Entity annotation specifies that the class is an entity and is mapped to a database table. The @Table annotation specifies the name of the database table to be used for mapping.

13. @RequestBody:

The @RequestBody annotation maps the HttpRequest body to a transfer or domain object, enabling automatic decentralization of the inbound HttpRequest body onto a Java object. Spring automatically desterilizes the JSON into a Java type, assuming an appropriate one is specified.

14. @ResponseStatus:

The @ResponseStatus marks a method or exception class with the status code and reason message that should be returned. The status code is applied to the HTTP response when the handler method is invoked, or whenever the specified exception is thrown.

15. @Exception Handler:

The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

16. @RestController:

The @RestController annotation in order to simplify the creation of RESTful web services. It's a convenient annotation that combines @Controller and @ResponseBody, which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.

17. @ControllerAdvice:

@ControllerAdvice is a specialization of the @Component annotation which allows to handle exceptions across the whole application in one global handling component. It can be viewed as an interceptor of exceptions thrown by methods annotated with @RequestMapping and similar.

18. @NotNull:

The **@NotNull** annotation is, actually, an explicit contract declaring that: A method should not return null. Variables (fields, local variables, and parameters) cannot hold a null value.

19. @NotBlank:

The @NotBlank annotation uses the NotBlank Validator class, which checks that a character sequence's trimmed length is not empty.

20. @Email

It is a most common use case to have EmailId as part of the API contract whenever it is designed for a user, and it is really important to validate this email-id as easily as possible.

21. @Length

It is the Hiberante-specific version of @size we can use either to validate the size of a field.

22. @JoinColumn

It is used to specify a column for joining an entity association or element collection.This annotation indicates that the enclosing entity is the owner of the relationship and the corresponding table has a foreign key column which references to the table of the non-owning side.

23. @SequenceGenerator :

The defines a primary key generator that may be referenced by name when a generator element is specified for the GeneratedValue annotation.

24. @Springbootapplication

It is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and componement scanning. It's same as declaring a class with

@Configuration,@EnableAutoConfiguration and @ComponentScan annotation.

Source Code

AdminController

```
package com.example.demo.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.entity.Admin;
import com.example.demo.error.AdminNotFoundException;
import com.example.demo.service.AdminService;
@RestController
public class AdminController {
@Autowired
AdminService adminService;
// insert
@PostMapping("/admins/")
public Admin saveAdmin(@RequestBody Admin admin)
{
return adminService.saveAdmin(admin);
```

```
}

//get Records

@GetMapping("/admins/")

public List<Admin> fetchAdminList()

{

return adminService.fetchAdminList();

}

//get the record by Id

@GetMapping("/admins/{adminId}")

public Admin fetchAdminById(@PathVariable("adminId") Long adminId) throws
AdminNotFoundException

{

return adminService.fetchAdminById(adminId);

}

//delete Records

@DeleteMapping("/admins/{adminId}")

public String deleteAdminById(@PathVariable("adminId") Long adminId) throws
AdminNotFoundException

{

adminService.deleteAdminById(adminId);

return "Admin is deleted";

}

//update Records

@PutMapping("/admins/{adminId}")

public Admin updateAdmin(@PathVariable ("adminId") Long adminId, @RequestBody Admin
admin) throws AdminNotFoundException
```

```
{
return adminService.updateAdmin(adminId,admin);
}
// get the name record
@GetMapping("/admins/name/{adminName}")
public Admin fetchAdminByName(@PathVariable("adminName") String adminName)
{
return adminService.fetchAdminByName(adminName);
}
// get the password record
@GetMapping("/admins/password/{adminPassword}")
public Admin fetchAdminByPassword(@PathVariable("adminPassword") String
adminPassword)
{
return adminService.fetchAdminByPassword(adminPassword);
}
// get the email id record
@GetMapping("/admins/emailid/{adminEmailId}")
public Admin fetchAdminByEmailid(@PathVariable("adminEmailId") String adminEmailId)
{
return adminService.fetchAdminByEmailId(adminEmailId);
}
// get the accomodationtype
@GetMapping("/admins/accomodationtype/{adminAccomodationType}")
public Admin fetchAdminByAccomodationType(@PathVariable("adminAccomodationType")
String adminAccomodationType)
```

```
{
return adminService.fetchAdminByAccomodationType(adminAccomodationType);
}
// get the package record
@GetMapping("/admins/package/{adminPackage}")
public Admin fetchAdminByPackage(@PathVariable("adminPackage") Float adminPackage)
{
return adminService.fetchAdminByPackage(adminPackage);
}
// get the room type
@GetMapping("/admins/roomtype/{adminRoomType}")
public Admin fetchAdminByRoomType(@PathVariable("adminRoomType") String
adminRoomType)
{
return adminService.fetchAdminByRoomType(adminRoomType);
}
}
```

Admin

```
package com.example.demo.entity;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```

```
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.SequenceGenerator;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import org.hibernate.validator.constraints.Length;
@Entity
public class Admin {
@Id
@GeneratedValue(generator="seq",strategy = GenerationType.AUTO)
@SequenceGenerator(name="seq",initialValue=1001)
private Long adminId;
@NotNull(message="Admin cannot be null")
@NotBlank(message="Admin name cannot be blank")
private String adminName;
@Length(min=4, max=10, message="Password cannot be less than 4 characters")
private String adminPassword;
@Email
private String adminEmailId;
@Length(min=3, message="AccomodationType cannot be less than 3 characters")
private String adminAccomodationType;
private Float adminPackage;
private String adminRoomType;
@OneToMany(targetEntity=Visitor.class,cascade=CascadeType.ALL)
```

```
@JoinColumn(name = "adminId")
private List<Visitor> visitors;
// generate the getter and setter method
public Long getAdminId() {
return adminId;
}
public void setAdminId(Long adminId) {
this.adminId = adminId;
}
public String getAdminName() {
return adminName;
}
public void setAdminName(String adminName) {
this.adminName = adminName;
}
public String getAdminPassword() {
return adminPassword;
}
public void setAdminPassword(String adminPassword) {
this.adminPassword = adminPassword;
}
public String getAdminEmailId() {
return adminEmailId;
}
public void setAdminEmailId(String adminEmailId) {
```

```
this.adminEmailId = adminEmailId;
}
public String getAdminAccomodationType() {
return adminAccomodationType;
}
public void setAdminAccomodationType(String adminAccomodationType) {
this.adminAccomodationType = adminAccomodationType;
}
public Float getAdminPackage() {
return adminPackage;
}
public void setAdminPackage(Float adminPackage) {
this.adminPackage = adminPackage;
}
public String getAdminRoomType() {
return adminRoomType;
}
public void setAdminRoomType(String adminRoomType) {
this.adminRoomType = adminRoomType;
}
public List<Visitor> getVisitors() {
return visitors;
}
public void setVisitors(List<Visitor> visitors) {
this.visitors = visitors;
```

```
}

// generate the to string

@Override

public String toString() {

return "Admin [adminId=" + adminId + ", adminName=" + adminName + ", adminPassword=" +
adminPassword+ ", adminEmailId=" + adminEmailId + ", adminAccomodationType=" +
adminAccomodationType+ ", adminPackage=" + adminPackage + ", adminRoomType=" +
adminRoomType + ", visitors=" + visitors+ "]";

}

// generate the constructor field(within the argument)

public Admin(Long adminId,

@NotNull(message = "Admin cannot be null") @NotBlank(message = "Admin name cannot be
blank") String adminName,

@Length(min = 4, max = 10, message = "Password cannot be less than 4 characters") String
adminPassword,

@Email String adminEmailId,

@Length(min = 3, message = "AccomodationType cannot be less than 3 characters") String
adminAccomodationType,

Float adminPackage, String adminRoomType, List<Visitor> visitors) {

super();

this.adminId = adminId;

this.adminName = adminName;

this.adminPassword = adminPassword;

this.adminEmailId = adminEmailId;

this.adminAccomodationType = adminAccomodationType;

this.adminPackage = adminPackage;

this.adminRoomType = adminRoomType;
```

```
this.visitors = visitors;
}
// generate the constructor superclass(without the argument)
public Admin() {
super();
}
}
```

AdminService

```
package com.example.demo.service;
import java.util.List;
import com.example.demo.entity.Admin;
import com.example.demo.error.AdminNotFoundException;
public interface AdminService {
Admin saveAdmin(Admin admin);
List<Admin> fetchAdminList();
Admin fetchAdminById(Long adminId) throws AdminNotFoundException;
void deleteAdminById(Long adminId) throws AdminNotFoundException;
Admin updateAdmin(Long adminId, Admin admin) throws AdminNotFoundException;
Admin fetchAdminByName(String adminName);
Admin fetchAdminByPassword(String adminPassword);
Admin fetchAdminByEmailId(String adminEmailId);
Admin fetchAdminByAccomodationType(String adminAccomodationType);
Admin fetchAdminByPackage(Float adminPackage);
Admin fetchAdminByRoomType(String adminRoomType);
}
```

AdminServiceImpl

```
package com.example.demo.service;

import java.util.List;

import java.util.Objects;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

import com.example.demo.entity.Admin;

import com.example.demo.error.AdminNotFoundException;

import com.example.demo.repository.AdminRepository;

@Service

public class AdminServiceImpl implements AdminService {

@Autowired

AdminRepository adminRepo;

// insert the record

@Override

public Admin saveAdmin(Admin admin)

{

return adminRepo.save(admin) ;

}

// display the all record

@Override

public List<Admin> fetchAdminList()

{

return adminRepo.findAll();
```

```
}
// display particular id
@Override
public Admin fetchAdminById(Long adminId) throws AdminNotFoundException
{
Optional<Admin> admin1= adminRepo.findById(adminId);//check in database
if(!admin1.isPresent()) {
throw new AdminNotFoundException("Admin not available");
}
return adminRepo.findById(adminId).get() ;
}
// delete the record
@Override
public void deleteAdminById(Long adminId) throws AdminNotFoundException {
Optional<Admin> admin1= adminRepo.findById(adminId);//check in database
if(!admin1.isPresent())
{
throw new AdminNotFoundException("Admin not available");
}
else
{
adminRepo.deleteById(adminId);
}
}
//update the record
```

```
@Override
public Admin updateAdmin(Long adminId, Admin admin) throws AdminNotFoundException {
Optional<Admin> admin1= adminRepo.findById(adminId);//check id
Admin admDB=null;
if(admin1.isPresent())
{
//id
admDB=adminRepo.findById(adminId).get();
//Name
if(Objects.nonNull(admin.getAdminName())&&!"".equalsIgnoreCase(admin.getAdminName()))
{
admDB.setAdminName(admin.getAdminName());
}
//Password
if(Objects.nonNull(admin.getAdminPassword())&&!"".equalsIgnoreCase(admin.getAdminPass
word())) {
admDB.setAdminPassword(admin.getAdminPassword());
System.out.println(admin.getAdminPassword());
}
//EmailId
if(Objects.nonNull(admin.getAdminEmailId())&&!"".equalsIgnoreCase(admin.getAdminEmailI
d())) {
admDB.setAdminEmailId(admin.getAdminEmailId());
System.out.println(admin.getAdminEmailId());
}
//AccomodationType
```

```
if(Objects.nonNull(admin.getAdminAccomodationType()) &&
!"".equalsIgnoreCase(admin.getAdminAccomodationType())) {

admDB.setAdminAccomodationType(admin.getAdminAccomodationType());

System.out.println(admin.getAdminAccomodationType());

}

//Package

if(Objects.nonNull(admin.getAdminPackage()) && !"".equals(admin.getAdminPackage())) {

admDB.setAdminPackage(admin.getAdminPackage());

System.out.println(admin.getAdminPackage());

}

//RoomType

if(Objects.nonNull(admin.getAdminRoomType()) &&
!"".equalsIgnoreCase(admin.getAdminRoomType())) {

admDB.setAdminRoomType(admin.getAdminRoomType());

System.out.println(admin.getAdminRoomType());

}

return adminRepo.save(admDB);

}//if

else

{

throw new AdminNotFoundException("Admin Not available");

}

}//update

//get the name

@Override

public Admin fetchAdminByName(String adminName)
```

```
{
return  adminRepo.findByAdminName(adminName);
}
// password
@Override
public Admin fetchAdminByPassword(String adminPassword)
{
return  adminRepo.findByAdminPassword(adminPassword);
}
//Emailid
@Override
public Admin fetchAdminByEmailId(String adminEmailId)
{
return adminRepo.findByAdminEmailId(adminEmailId) ;
}
//AccomodationType
@Override
public Admin fetchAdminByAccomodationType(String adminAccomodationType)
{
return adminRepo.findByAdminAccomodationType(adminAccomodationType);
}
//Package
@Override
public Admin fetchAdminByPackage(Float adminPackage)
{
```

```
return adminRepo.findByAdminPackage(adminPackage);
}
// RoomType
@Override
public Admin fetchAdminByRoomType(String adminRoomType)
{
return adminRepo.findByAdminRoomType(adminRoomType) ;
}
}
```

AdminRepository

```
package com.example.demo.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.entity.Admin;
@Repository
public interface AdminRepository extends JpaRepository<Admin, Long>{
public Admin findByAdminName(String adminName);
public Admin findByAdminPassword(String adminPassword);
public Admin findByAdminEmailId(String adminEmailId);
public Admin findByAdminAccomodationType(String adminAccomodationType);
public Admin findByAdminPackage(Float adminPackage);
public Admin findByAdminRoomType(String adminRoomType);
}
```

VisitorController

```
package com.example.demo.controller;
```

```
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.entity.Visitor;
import com.example.demo.error.VisitorNotFoundException;
import com.example.demo.service.VisitorService;
@RestController
public class VisitorController {
@Autowired
VisitorService visitorService;
// insert the record
@PostMapping("/visitors/")
public Visitor saveVisitor(@RequestBody Visitor visitor)
{
return visitorService.saveVisitor(visitor);
}
// display the all record
@GetMapping("/visitors/")
public List<Visitor> fetchVisitorList()
```

```
{
return visitorService.fetchVisitorList();
}
//display the particular id
@GetMapping("/visitors/{visitorId}")
public Visitor fetchVisitorById(@PathVariable("visitorId") Long visitorId) throws
VisitorNotFoundException
{
return visitorService.fetchVisitorById(visitorId);
}
// delete the record
@DeleteMapping("/visitors/{visitorId}")
public String deleteVisitorById(@PathVariable("visitorId") Long visitorId) throws
VisitorNotFoundException
{
visitorService.deleteVisitorById(visitorId);
return "Visitor is deleted";
}
//Update the record
@PutMapping("/visitors/{visitorId}")
public Visitor updateVisitor(@PathVariable ("visitorId") Long visitorId, @RequestBody Visitor
visitor) throws VisitorNotFoundException
{
return visitorService.updateVisitor(visitorId,visitor);
}
//get name record
```

```
@GetMapping("/visitors/name/{visitorName}")
public Visitor fetchVisitorByName(@PathVariable("visitorName") String visitorName)
{
return visitorService.fetchVisitorByName(visitorName);
}
//get the Mobilenumber
@GetMapping("/visitors/mobile/{visitorMobileNumber}")
public Visitor fetchVisitorByMobileNumber(@PathVariable("visitorMobileNumber") String
visitorMobileNumber)
{
return visitorService.fetchVisitorByMobileNumber(visitorMobileNumber);
}
// get the emailid
@GetMapping("/visitors/emailid/{visitorEmailId}")
public Visitor fetchVisitorByEmailId(@PathVariable("visitorEmailId") String visitorEmailId)
{
return visitorService.fetchVisitorByEmailId(visitorEmailId);
}
// get the address
@GetMapping("/visitors/address/{visitorAddress}")
public Visitor fetchVisitorByAddress(@PathVariable("visitorAddress") String visitorAddress)
{
return visitorService.fetchVisitorByAddress(visitorAddress);
}
// get the destination
```

```
@GetMapping("/visitors/destination/{visitorDestination}")
public Visitor fetchVisitorByDestination(@PathVariable("visitorDestination") String
visitorDestination)
{
return visitorService.fetchVisitorByDestination(visitorDestination);
}
}
```

Visitor

```
package com.example.demo.entity;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import org.hibernate.validator.constraints.Length;
@Entity
public class Visitor {
@Id
@GeneratedValue
private Long visitorId;
@NotNull(message="Name cannot be null")
@NotBlank(message="Visitor name cannot be blank")
private String visitorName;
@Length(min=10, max=13, message="Mobile number cannot be less than 10 characters")
```

```
private String visitorMobileNumber;
@Email
private String visitorEmailId;
@Length(min=3, message="Address cannot be less than 3 characters")
private String visitorAddress;
@NotNull(message="Destination  cannot be null")
@NotBlank(message="Visitor Destination name cannot be blank")
private String visitorDestination;
// generate the setter and getter method
public Long getVisitorId() {
return visitorId;
}
public void setVisitorId(Long visitorId) {
this.visitorId = visitorId;
}
public String getVisitorName() {
return visitorName;
}
public void setVisitorName(String visitorName) {
this.visitorName = visitorName;
}
public String getVisitorMobileNumber() {
return visitorMobileNumber;
}
public void setVisitorMobileNumber(String visitorMobileNumber) {
```

```
this.visitorMobileNumber = visitorMobileNumber;
}
public String getVisitorEmailId() {
return visitorEmailId;
}
public void setVisitorEmailId(String visitorEmailId) {
this.visitorEmailId = visitorEmailId;
}
public String getVisitorAddress() {
return visitorAddress;
}
public void setVisitorAddress(String visitorAddress) {
this.visitorAddress = visitorAddress;
}
public String getVisitorDestination() {
return visitorDestination;
}
public void setVisitorDestination(String visitorDestination) {
this.visitorDestination = visitorDestination;
}
// generate the to string
@Override
public String toString() {
return "Visitor [visitorId=" + visitorId + ",visitorName="+visitorName+
",visitorMobileNumber="+ visitorMobileNumber + ", visitorEmailId=" + visitorEmailId + ",
visitorAddress=" + visitorAddress+ ", visitorDestination=" + visitorDestination + "]";
```

```
}

// generate the constructor field (within the argument)

public Visitor(Long visitorId,

@NotNull(message = "Name cannot be null") @NotBlank(message = "Visitor name cannot be blank") String visitorName,

@Length(min = 10, max = 13, message = "Mobile number cannot be less than 10 characters") String visitorMobileNumber,

@Email String visitorEmailId,

@Length(min = 3, message = "Address cannot be less than 3 characters") String visitorAddress,

@NotNull(message = "Destination  cannot be null") @NotBlank(message = "Visitor Destination name cannot be blank") String visitorDestination) {

super();

this.visitorId = visitorId;

this.visitorName = visitorName;

this.visitorMobileNumber = visitorMobileNumber;

this.visitorEmailId = visitorEmailId;

this.visitorAddress = visitorAddress;

this.visitorDestination = visitorDestination;

}

// generate the constructor superclass(without the argument)

public Visitor() {

super();

}

}
```

VisitorService

```
package com.example.demo.service;
```

```
import java.util.List;
import com.example.demo.entity.Visitor;
import com.example.demo.error.VisitorNotFoundException;
public interface VisitorService {
Visitor saveVisitor(Visitor visitor);
List<Visitor> fetchVisitorList();
Visitor fetchVisitorById(Long visitorId) throws VisitorNotFoundException;
void deleteVisitorById(Long visitorId) throws VisitorNotFoundException;
Visitor updateVisitor(Long visitorId, Visitor visitor) throws VisitorNotFoundException;
Visitor fetchVisitorByName(String visitorName);
Visitor fetchVisitorByMobileNumber(String visitorMobileNumber);
Visitor fetchVisitorByEmailId(String visitorEmailId);
Visitor fetchVisitorByAddress(String visitorAddress);
Visitor fetchVisitorByDestination(String visitorDestination);
}
```

VisitorServiceImpl

```
package com.example.demo.service;
import java.util.List;
import java.util.Objects;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.demo.entity.Visitor;
import com.example.demo.error.VisitorNotFoundException;
import com.example.demo.repository.VisitorRepository;
```

```
@Service
public class VisitorServiceImpl implements VisitorService {
@Autowired
VisitorRepository visitorRepo;
// insert the record
@Override
public Visitor saveVisitor(Visitor visitor)
{
return visitorRepo.save(visitor) ;
}
//display the all record
@Override
public List<Visitor> fetchVisitorList()
{
return visitorRepo.findAll();
}
// display the Id
@Override
public Visitor fetchVisitorById(Long visitorId) throws VisitorNotFoundException
{
Optional<Visitor> visitor1= visitorRepo.findById(visitorId);//check in database
if(!visitor1.isPresent()) {
throw new VisitorNotFoundException("Visitor not available");
}
return visitorRepo.findById(visitorId).get() ;
```

```
}
// delete the record
@Override
public void deleteVisitorById(Long visitorId) throws VisitorNotFoundException
{
Optional<Visitor> visitor1= visitorRepo.findById(visitorId);//check in database
if(!visitor1.isPresent()) {
throw new VisitorNotFoundException("Visitor not available");
}
else {
visitorRepo.deleteById(visitorId);
}
}
// update the record
@Override
public Visitor updateVisitor(Long visitorId, Visitor visitor) throws VisitorNotFoundException
{
Optional<Visitor> visitor1= visitorRepo.findById(visitorId);//check id
Visitor visDB=null;
if(visitor1.isPresent())
{
// id
visDB=visitorRepo.findById(visitorId).get();
//Name
```

```
if(Objects.nonNull(visitor.getVisitorName())&&!"".equalsIgnoreCase(visitor.getVisitorName()))
{

visDB.setVisitorName(visitor.getVisitorName());

}

// MobileNumber

if(Objects.nonNull(visitor.getVisitorMobileNumber())&&!"".equalsIgnoreCase(visitor.getVisito
rMobileNumber())) {

visDB.setVisitorMobileNumber(visitor.getVisitorMobileNumber());

System.out.println(visitor.getVisitorMobileNumber());

}

//MailId

if(Objects.nonNull(visitor.getVisitorEmailId())&&!"".equalsIgnoreCase(visitor.getVisitorEmailI
d())) {

visDB.setVisitorEmailId(visitor.getVisitorEmailId());

System.out.println(visitor.getVisitorEmailId());

}

//Address

if(Objects.nonNull(visitor.getVisitorAddress())&&!"".equalsIgnoreCase(visitor.getVisitorAddre
ss())) {

visDB.setVisitorAddress(visitor.getVisitorAddress());

System.out.println(visitor.getVisitorAddress());

}

//Destination

if(Objects.nonNull(visitor.getVisitorDestination())&&!"".equalsIgnoreCase(visitor.getVisitorDe
stination())) {

visDB.setVisitorDestination(visitor.getVisitorDestination());

System.out.println(visitor.getVisitorDestination());
```

```
}
return visitorRepo.save(visDB) ;
}//if
else
{
throw new VisitorNotFoundException("Visitor Not available");
}
}//update
//visitor name
@Override
public Visitor fetchVisitorByName(String visitorName)
{
return visitorRepo.findByVisitorName(visitorName);
}
// visitor MobileNumber
@Override
public Visitor fetchVisitorByMobileNumber(String visitorMobileNumber)
{
return visitorRepo.findByVisitorMobileNumber(visitorMobileNumber);
}
// visitor Emailid
@Override
public Visitor fetchVisitorByEmailId(String visitorEmailId)
{
return visitorRepo.findByVisitorEmailId(visitorEmailId);
```

```
}
// visitor Address
@Override
public Visitor fetchVisitorByAddress(String visitorAddress)
{
return visitorRepo.findByVisitorAddress(visitorAddress) ;
}
// visitor destination
@Override
public Visitor fetchVisitorByDestination(String visitorDestination)
{
return visitorRepo.findByVisitorDestination(visitorDestination);
}
}
```

VisitorRepository

```
package com.example.demo.repository;
import  org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.entity.Visitor;
@Repository
public interface VisitorRepository extends JpaRepository<Visitor, Long>{
public Visitor findByVisitorName(String visitorName);
public Visitor findByVisitorMobileNumber(String visitorMobileNumber);
public Visitor findByVisitorEmailId(String visitorEmailId);
public Visitor findByVisitorAddress(String visitorAddress);
```

```
public Visitor findByVisitorDestination(String visitorDestination);
}
```

Error Message

```
package com.example.demo.entity;
import org.springframework.http.HttpStatus;
public class ErrorMessage {
private HttpStatus status;
private String message;
// generate the setter and getter method
public HttpStatus getStatus() {
return status;
}
public void setStatus(HttpStatus status) {
this.status = status;
}
public String getMessage() {
return message;
}
public void setMessage(String message) {
this.message = message;
}
// generate the to string
@Override
public String toString() {
return "ErrorMessage [status=" + status + ", message=" + message + "]";
```

```
}
// generate the constructor(within the argument)
public ErrorMessage(HttpStatus status, String message) {
super();
this.status = status;
this.message = message;
}
// generate the constructor(without the argument)
public ErrorMessage() {
super();
}
}
```

AdminNotFoundException

```
package com.example.demo.error;
public class AdminNotFoundException extends Exception {
private static final long serialversionUID=1L;
public AdminNotFoundException(String s)
{
super(s);
}
}
```

VisitorNotFoundException

```
package com.example.demo.error;
public class VisitorNotFoundException extends Exception{
private static final long serialversionUID=1L;
```

```
public VisitorNotFoundException(String s)
{
super(s);
}
}
```

RestResponseEntityExceptionHandler

```
package com.example.demo.error;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
import com.example.demo.entity.ErrorMessage;
@ControllerAdvice
@ResponseStatus
public class RestResponseEntityExceptionHandler extends ResponseEntityExceptionHandler {
// Admin
@ExceptionHandler(AdminNotFoundException.class)
public ResponseEntity<ErrorMessage> AdminNotFoundException(AdminNotFoundException
exception,WebRequest request)
{
ErrorMessage message=new
ErrorMessage(HttpStatus.NOT_FOUND,exception.getMessage());//constructor
```

```
return ResponseEntity.status(HttpStatus.NOT_FOUND).body(message);

}

// visitor

@ExceptionHandler(VisitorNotFoundException.class)

public ResponseEntity<ErrorMessage> VisitorNotFoundException(VisitorNotFoundException exception,WebRequest request)

{

ErrorMessage message=new ErrorMessage(HttpStatus.NOT_FOUND,exception.getMessage());//constructor

return ResponseEntity.status(HttpStatus.NOT_FOUND).body(message);

}

}
```

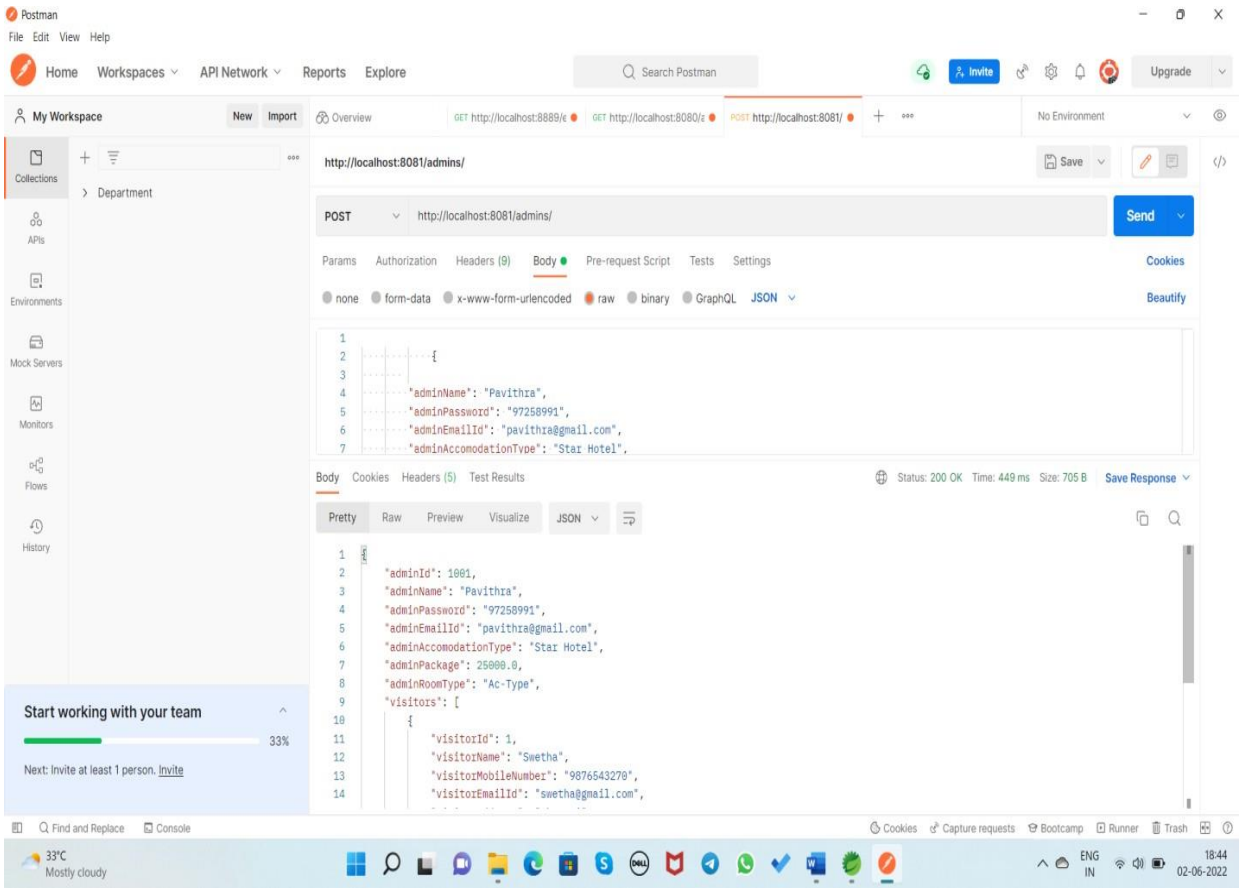
ApplicationProperties

```
server.port = 8081

spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver

spring.datasource.url = jdbc:mysql://localhost:3306/tourism

spring.datasource.username = root

spring.datasource.password = root

spring.jpa.show-sql = true

spring.jpa.generate-ddl= true

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto property

spring.jpa.hibernate.ddl-auto=create
```

Screenshot

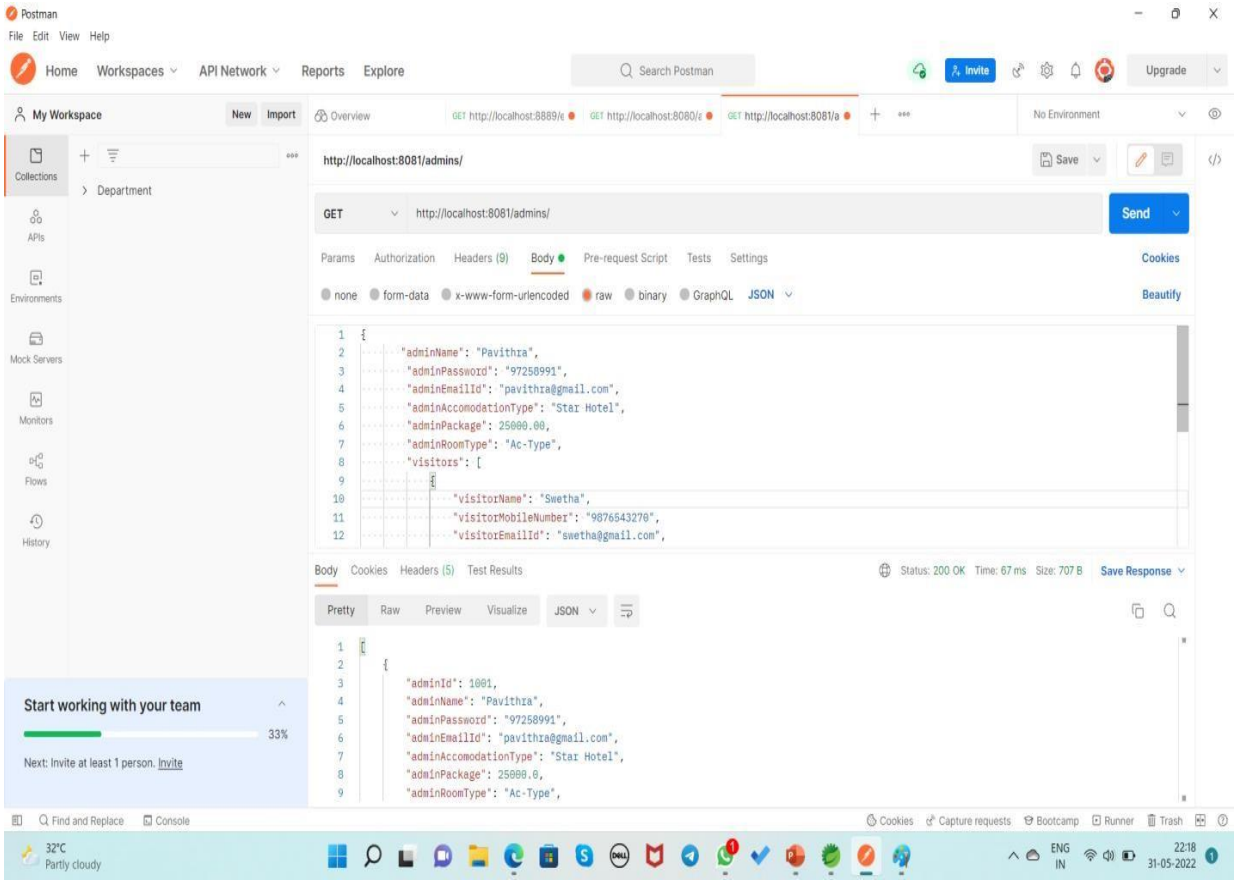
Step 1: Insert **Admin and Visitor** Record By Using **POST** Method

URL: **http://localhost:8081/admins/**



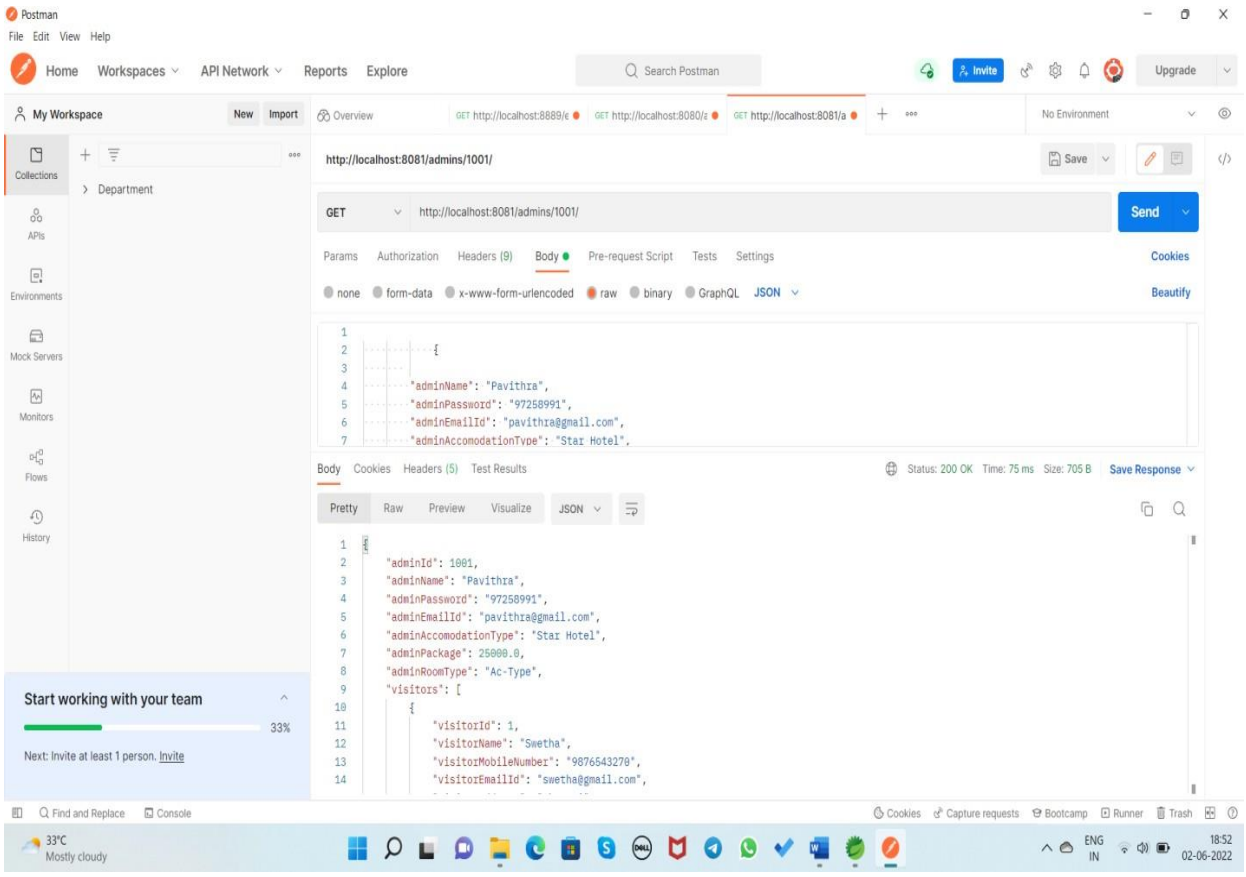
Step 2 : Display the all record of **Admin** By Using **GET** Method

URL : http://localhost:8081/admins/



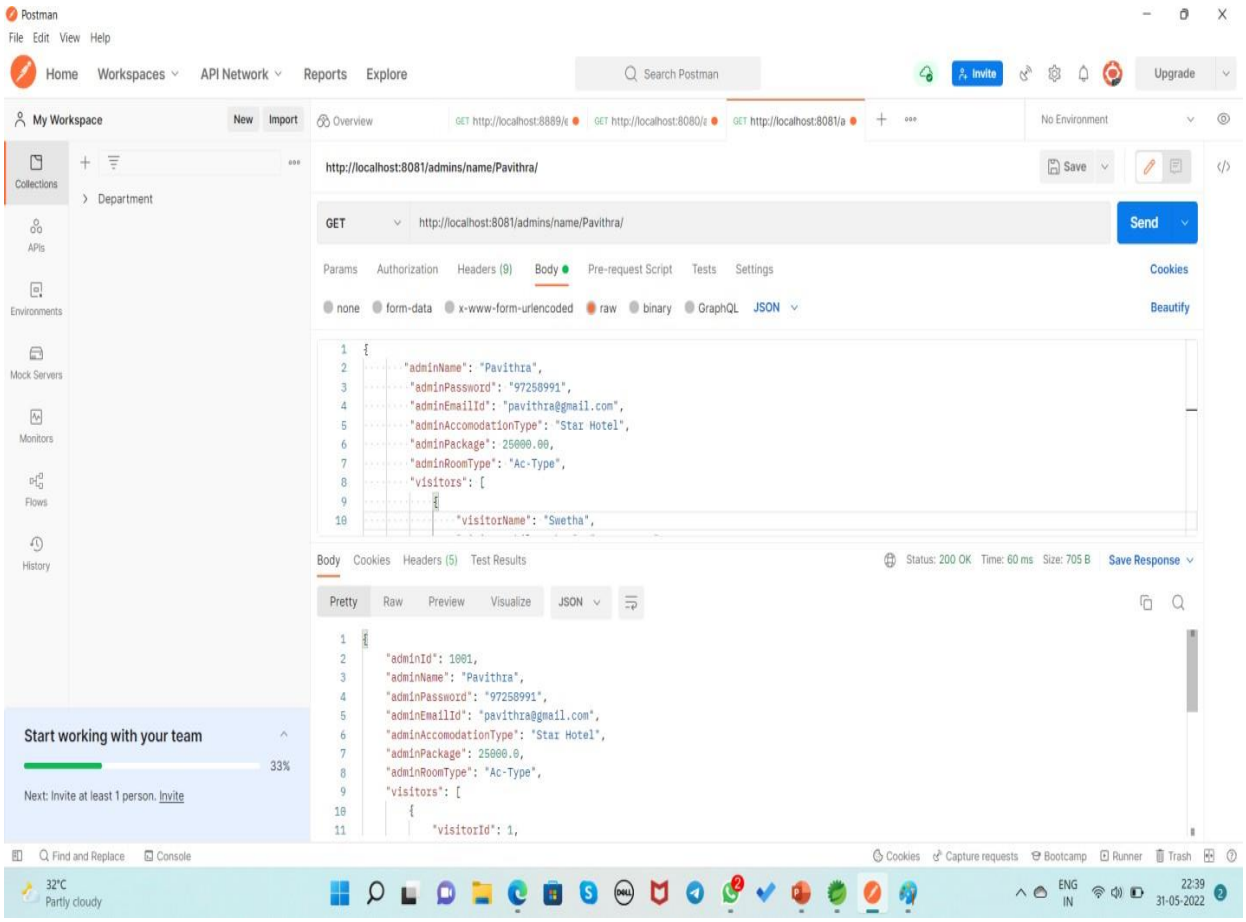
Step 3: Display the record of **Admin** based on **Id** By Using **GET** Method

URL : http://localhost:8081/admins/1001/



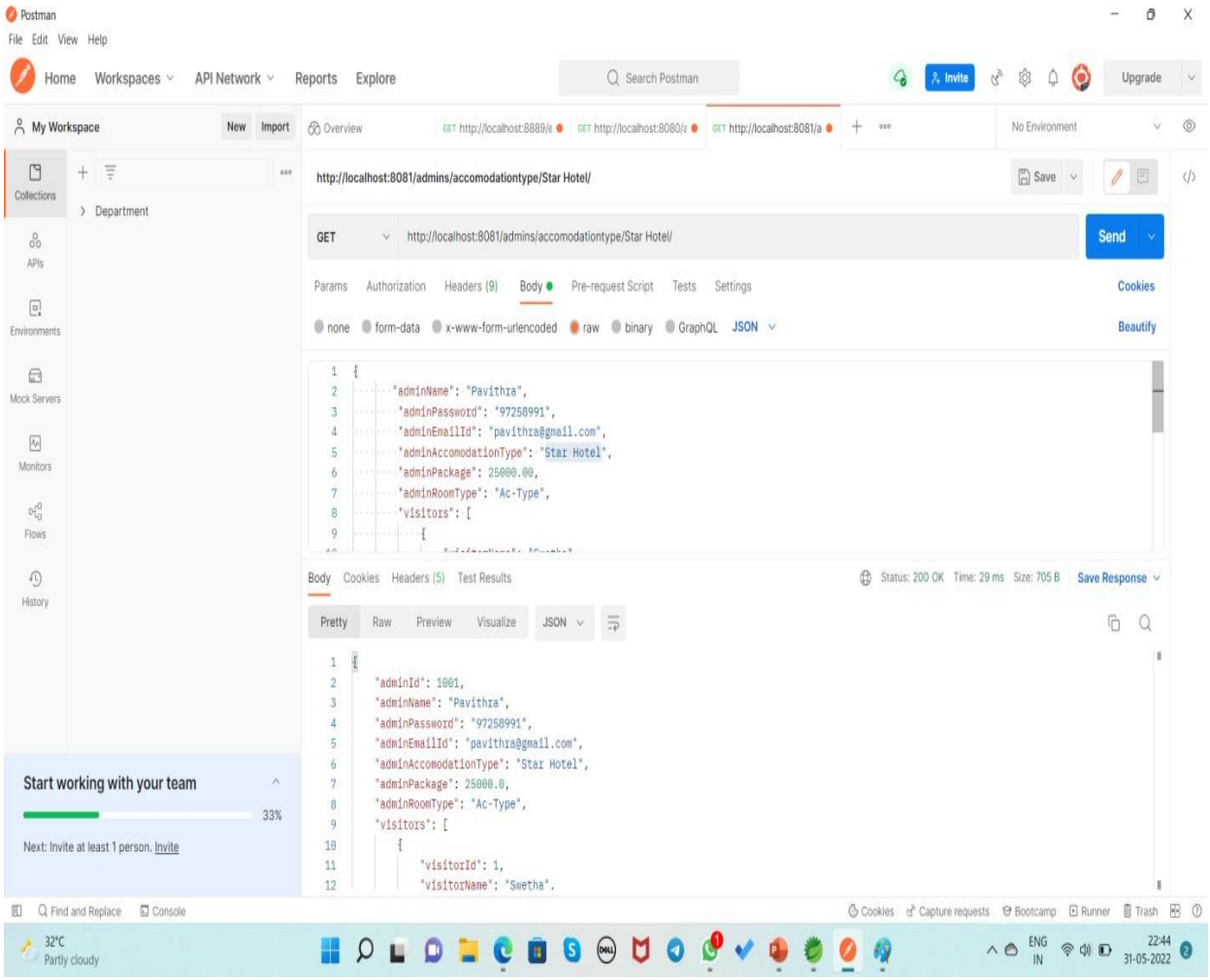
Step 4: Display the record of**Admin**based on**Name** By Using**GET**Method

URL : http://localhost:8081/admins/name/Pavithra/



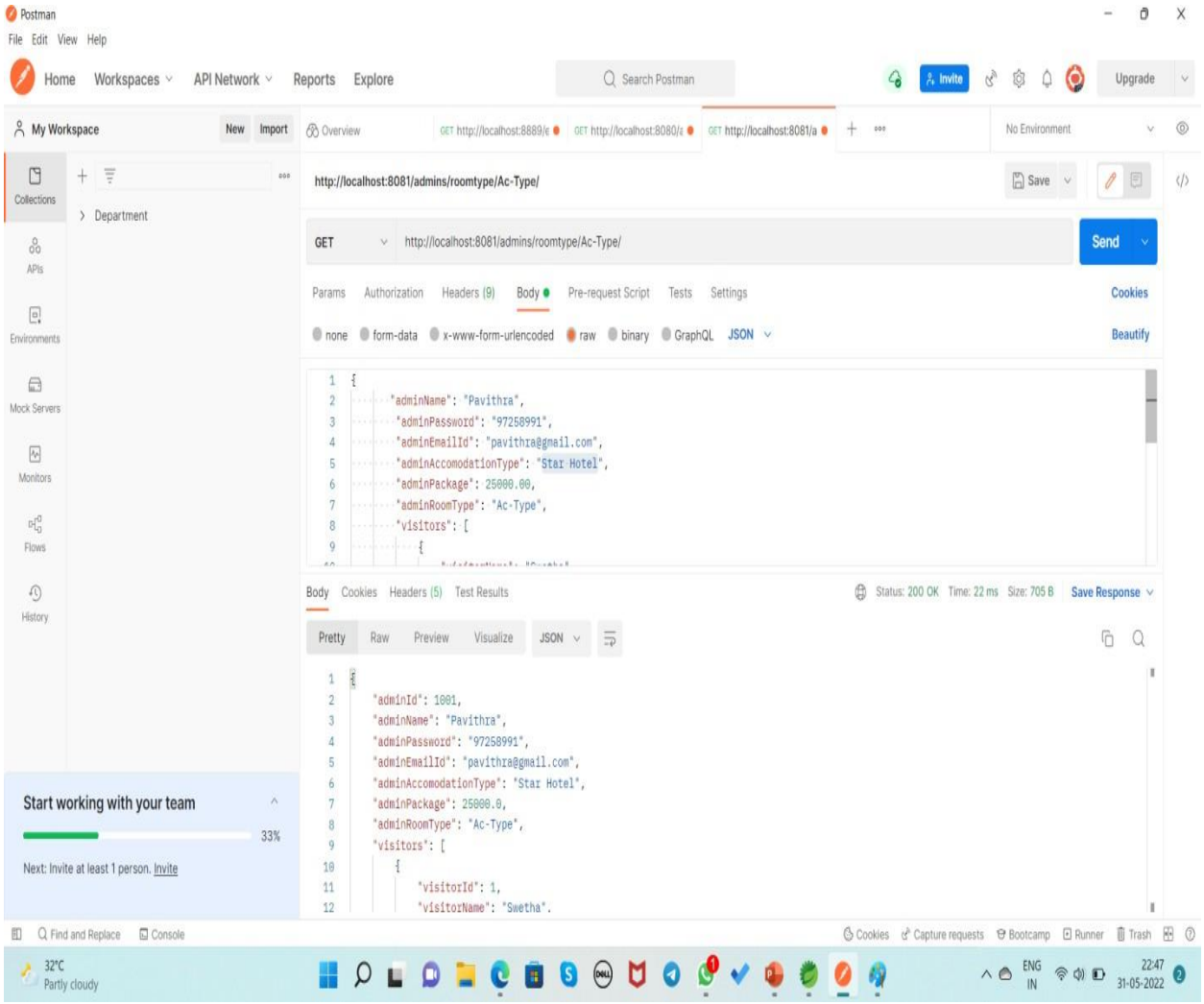
Step 5: Display the record of **Admin** based on **AccomodationType** By Using **GET** Method

URL : http://localhost:8081/admins/accomodationtype/Star Hotel/



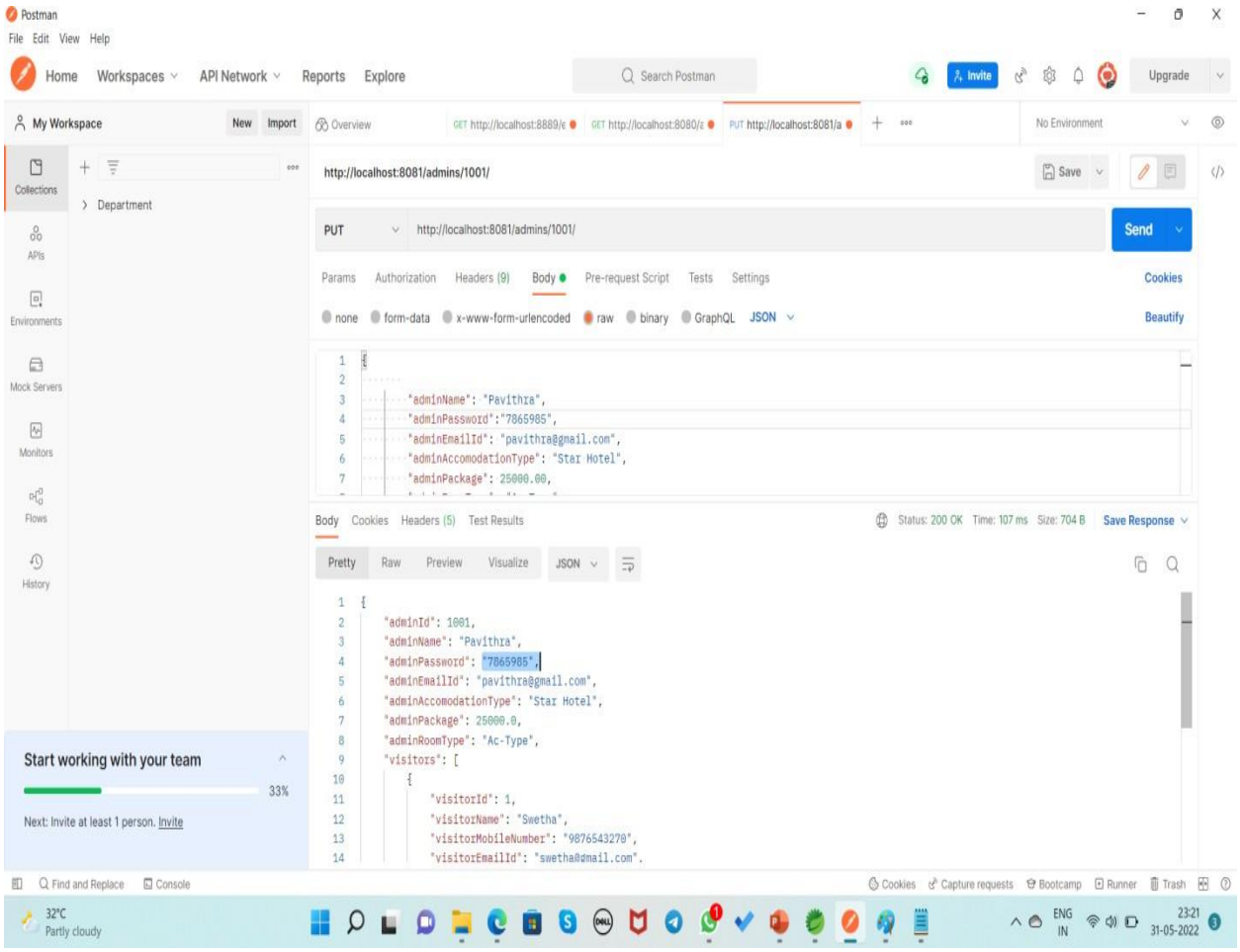
Step 6 : Display the record of **Admin** based on **RoomType** By Using **GET** Method

URL :[http://localhost:8081/admins/room type/Ac-Type/](http://localhost:8081/admins/room%20type/Ac-Type/)



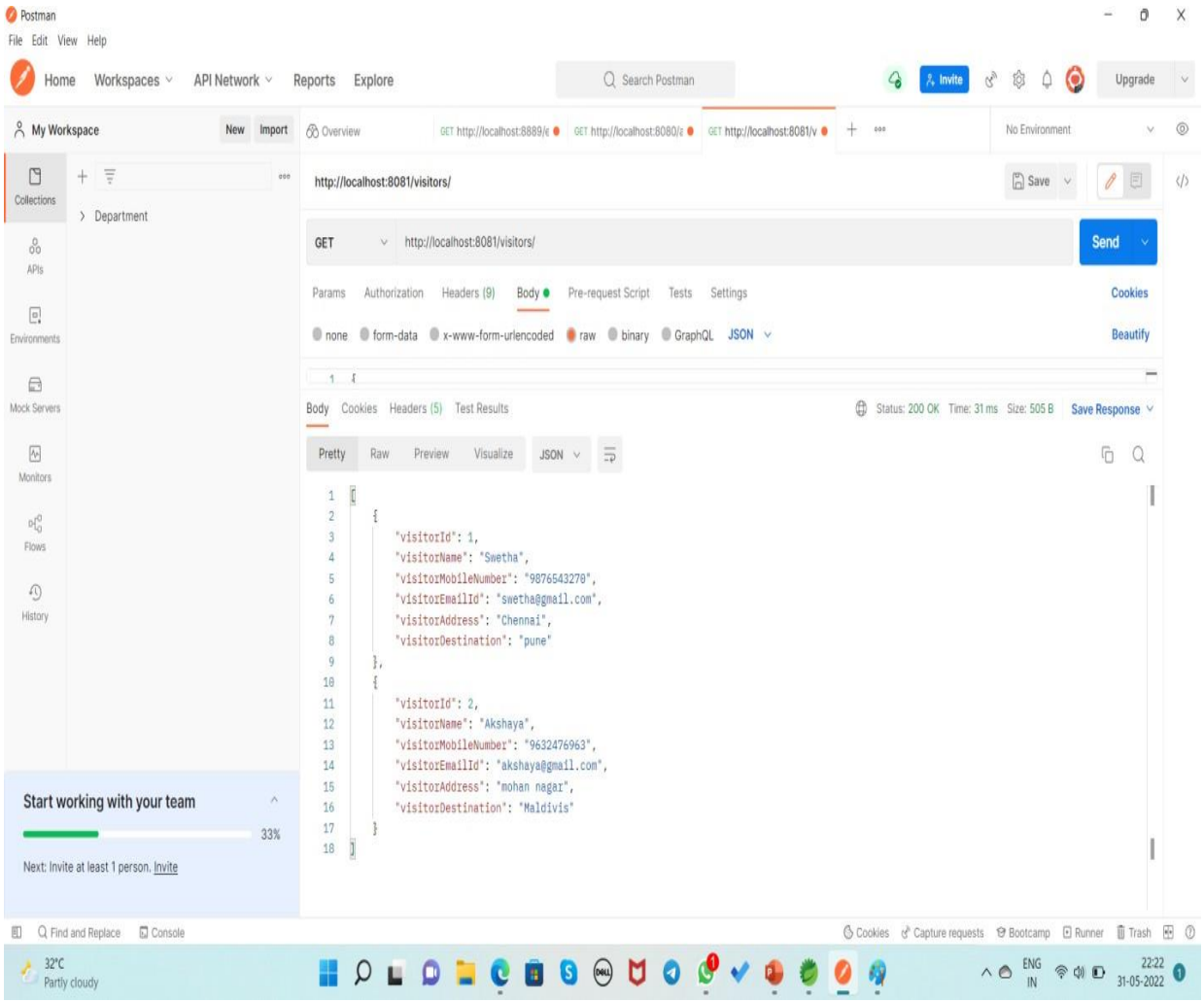
Step 7: If **Admin** wants to change the **Password** By Using **PUT** Method.

URL : http://localhost:8081/admins/1001/



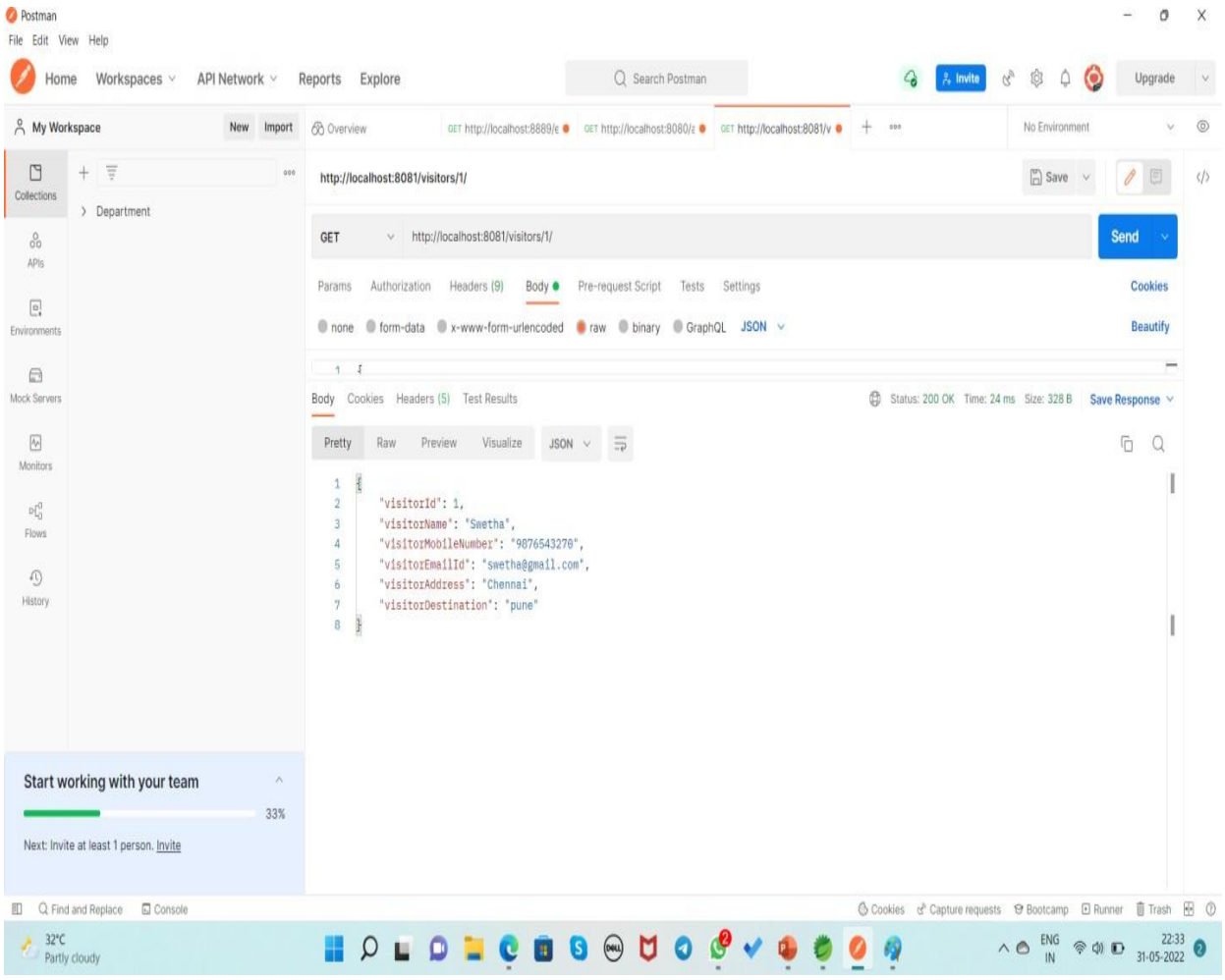
Step 8: Display the all record of **Visitor** By Using **GET** Method

URL : http://localhost:8081/visitors/



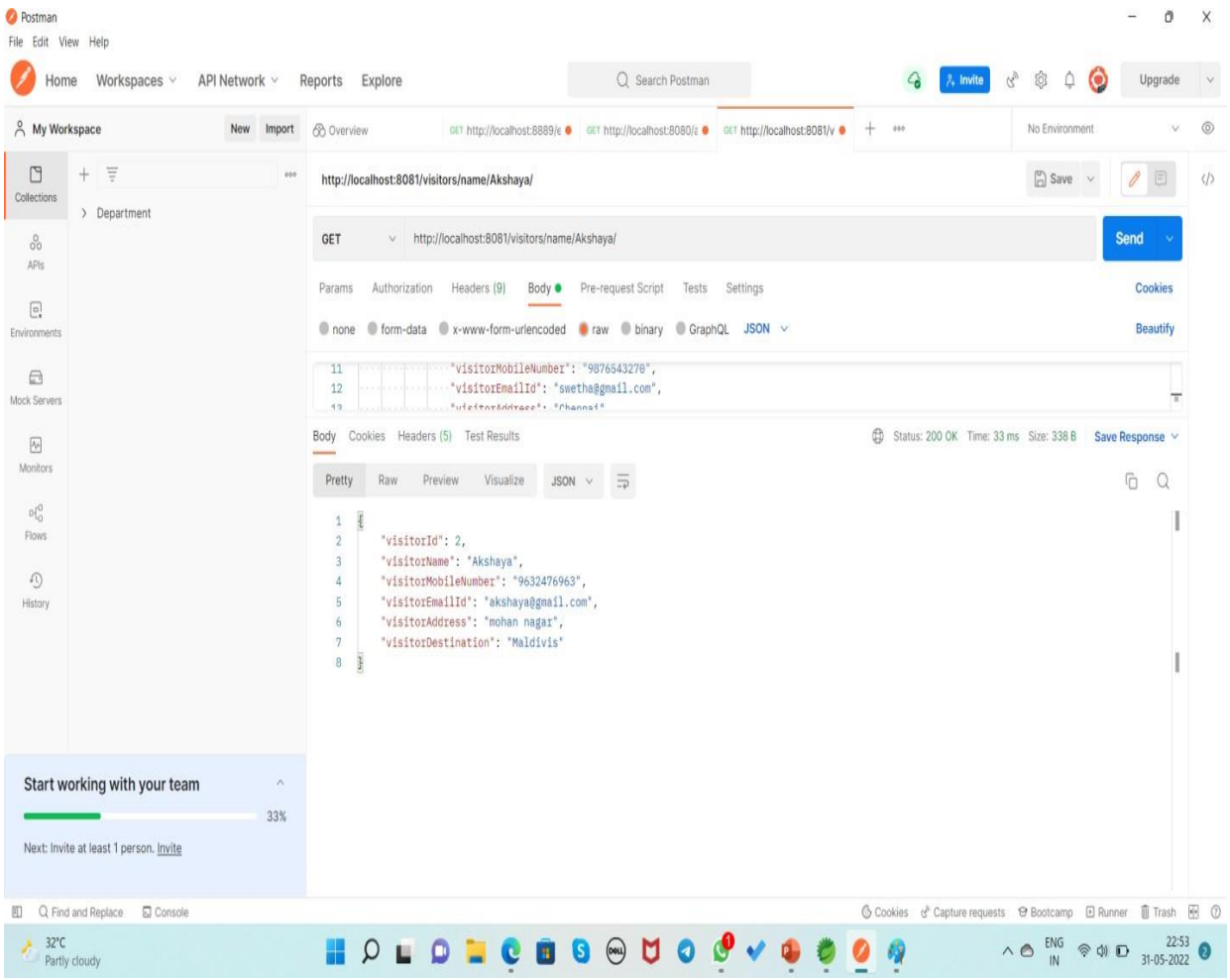
Step 9 : Display the record of **Visitor** based on **Id** By Using **GET** Method

URL : http://localhost:8081/visitors/1/



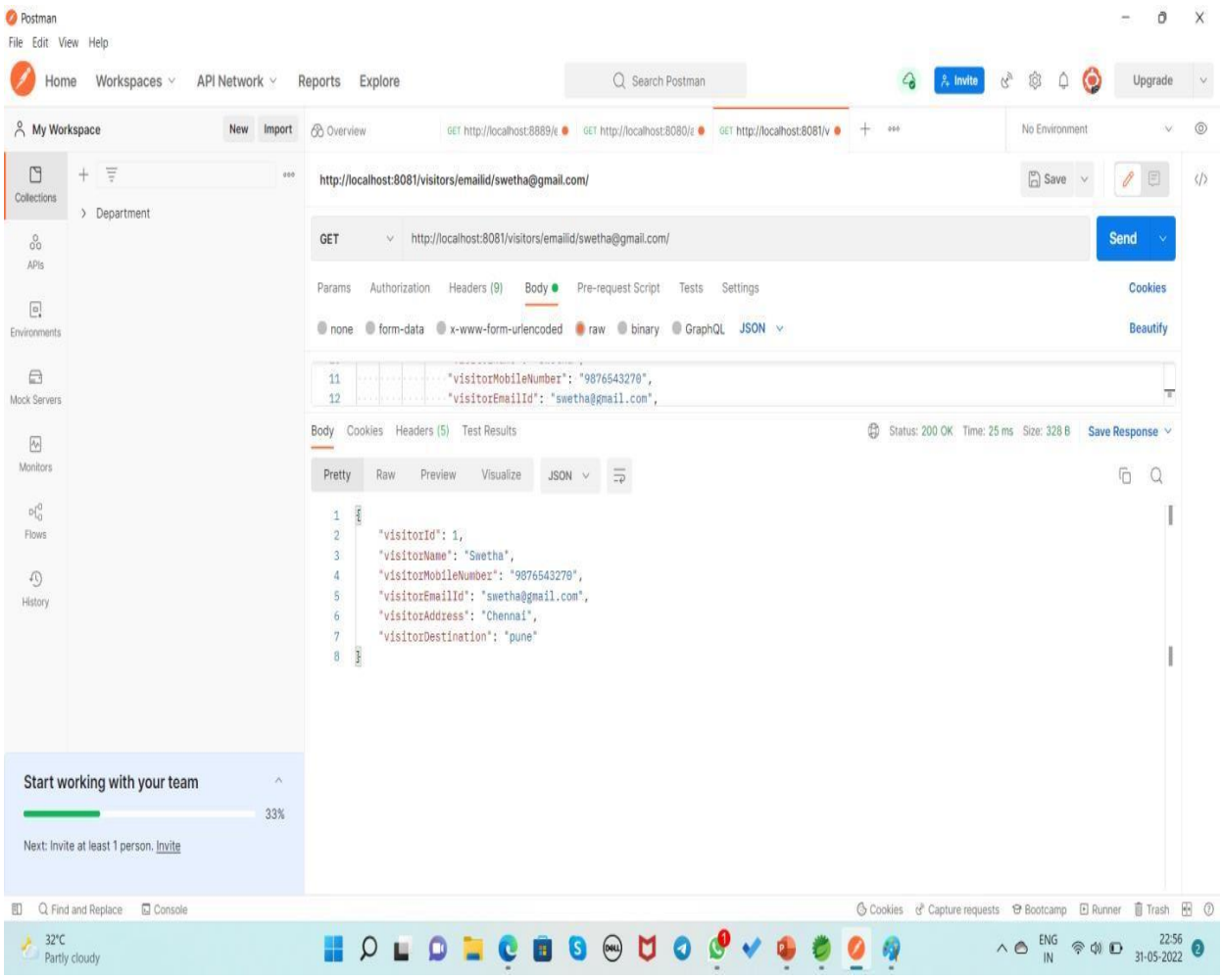
Step 10 : Display the record of **Visitor** based on **Name** By Using **GET** Method

URL : http://localhost:8081/visitors/name/Akshaya/



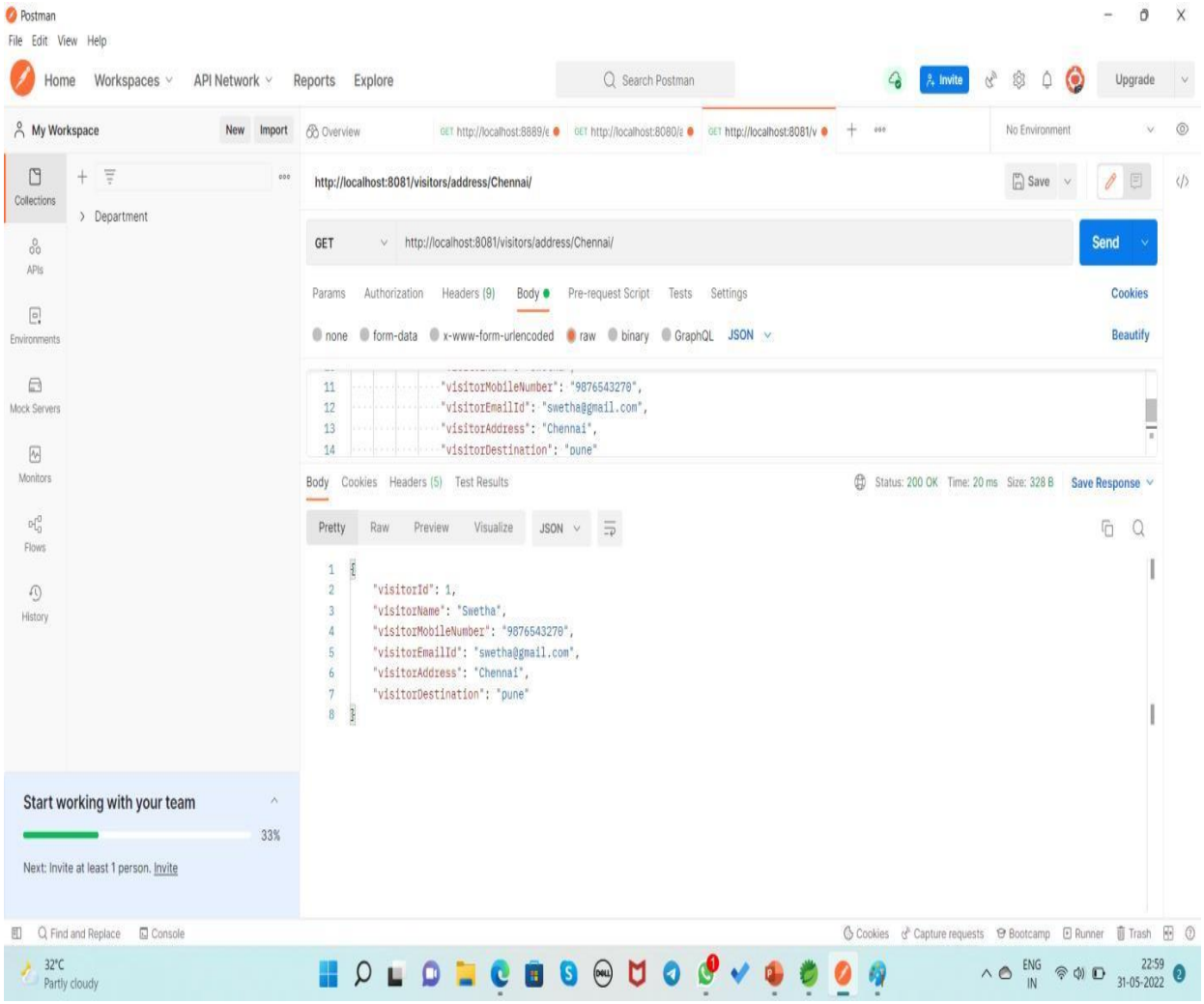
Step 11: Display the record of **Visitor** based on **EmailId** By Using **GET** Method

URL : **http://localhost:8081/visitors/emailid/swetha@gmail.com/**



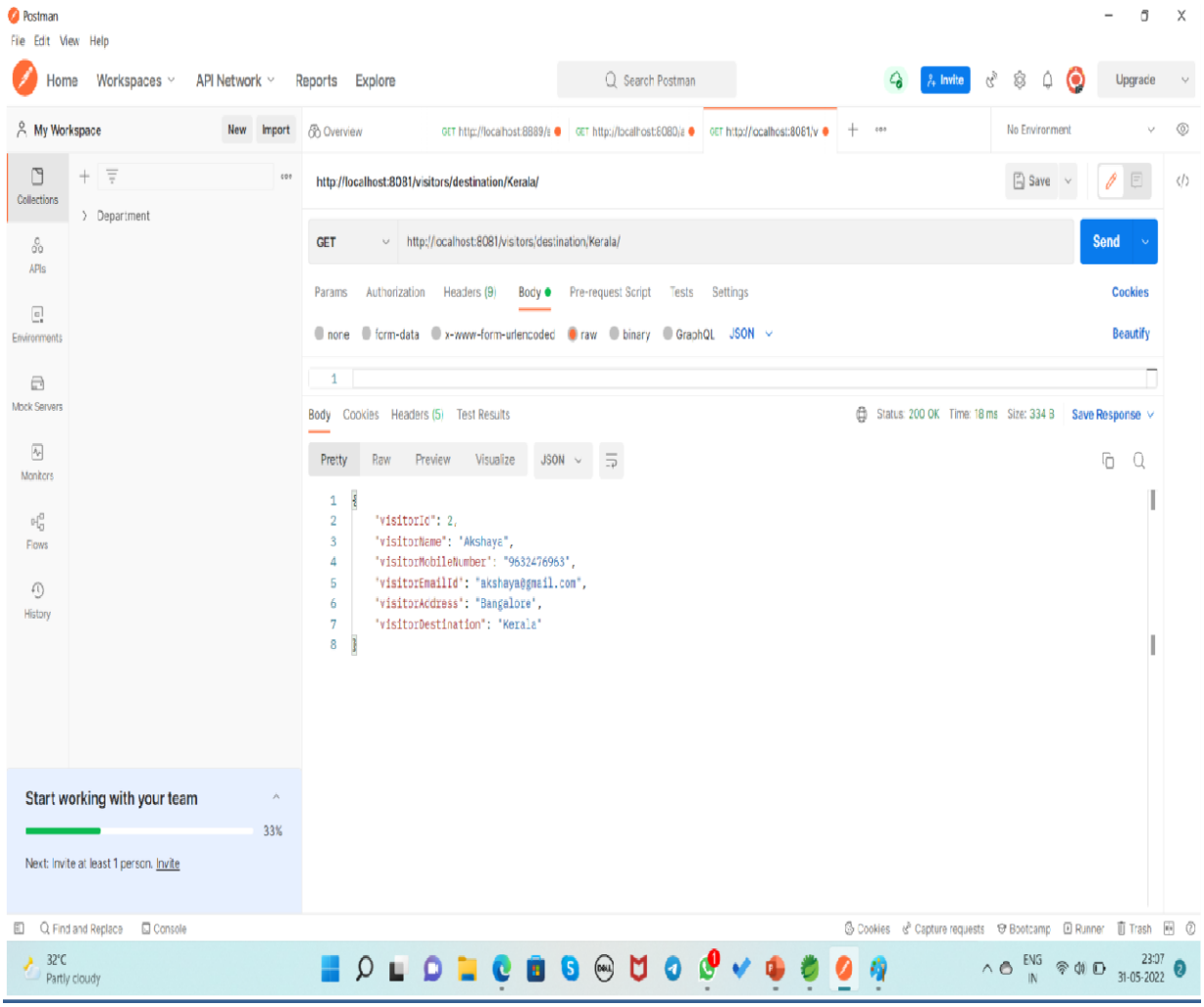
Step 12: Display the record of **Visitor** based on **Address** By Using **GET** Method

URL : http://localhost:8081/visitors/address/Chennai/



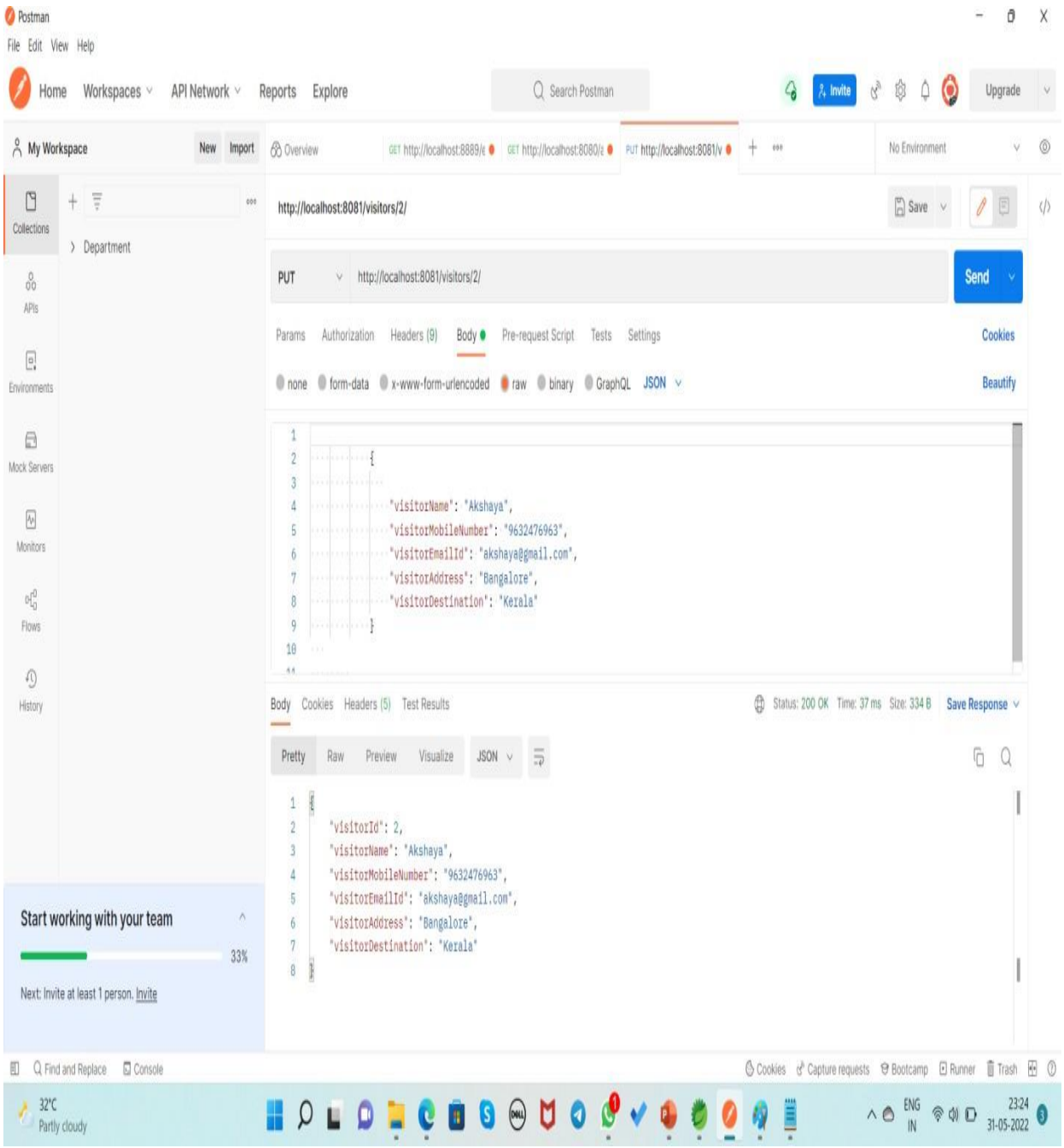
Step13: Display the record of **Visitor** based on **Destination** By Using **GET** Method

URL : http://localhost:8081/visitors/destination/Kerala/



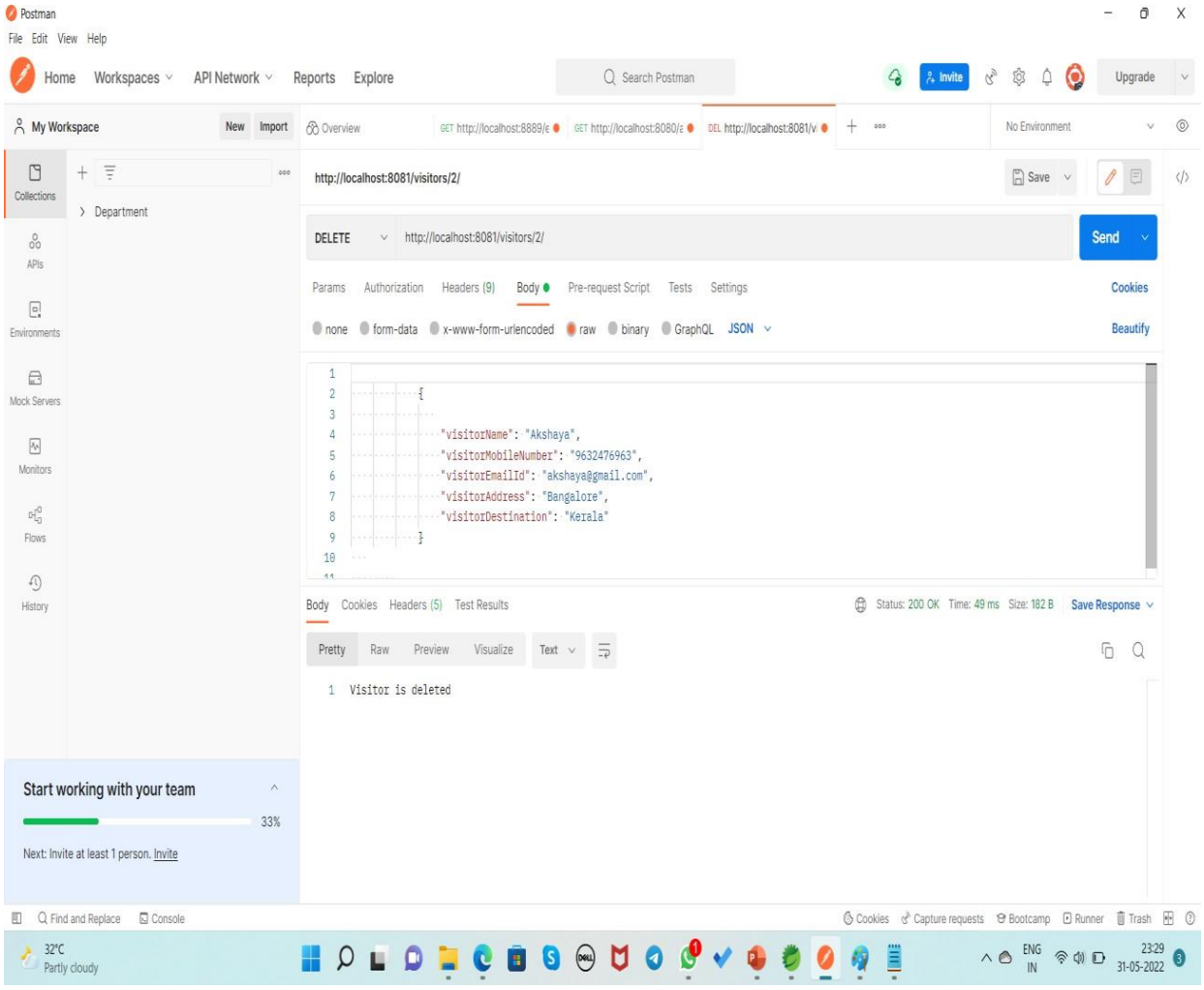
Step 14: If **Visitor** wants to change the **Address and Destination** By Using **PUT** Method

URL : http://localhost:8081/visitors/2/



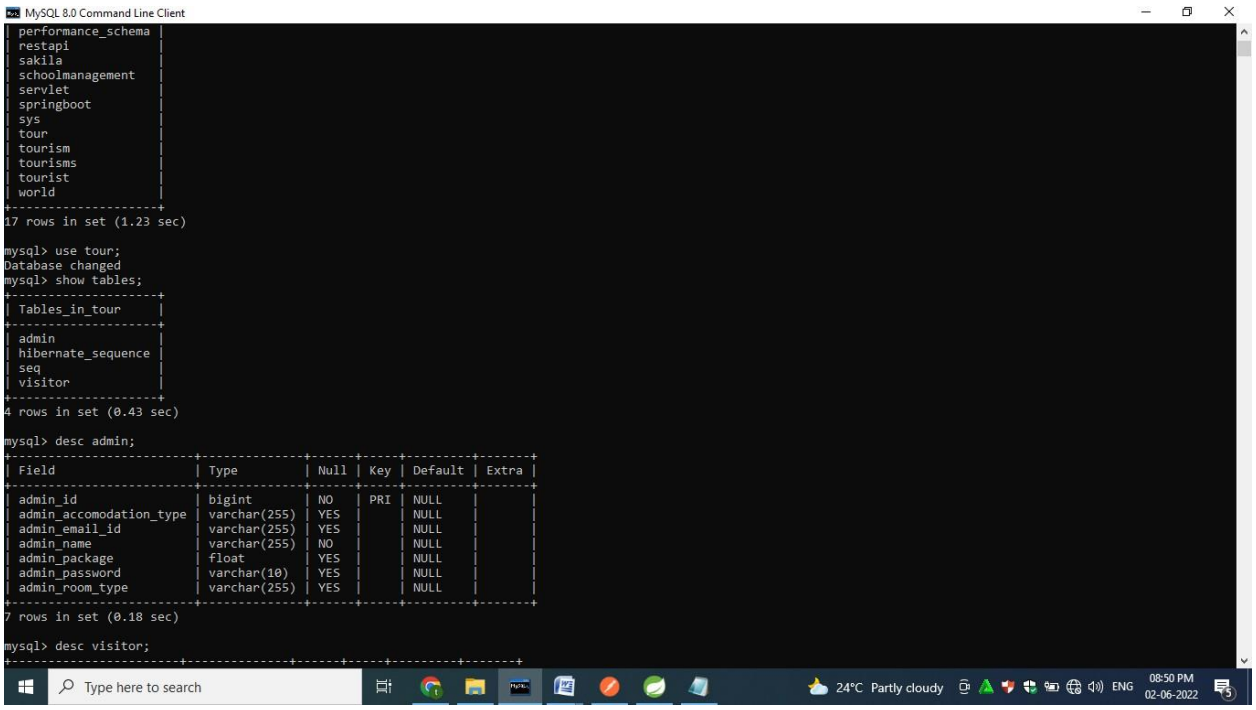
Step 15: Deleting the **Visitor** Record Based on **Id**By Using **DELETE**Method

URL : http://localhost:8081/visitors/2/



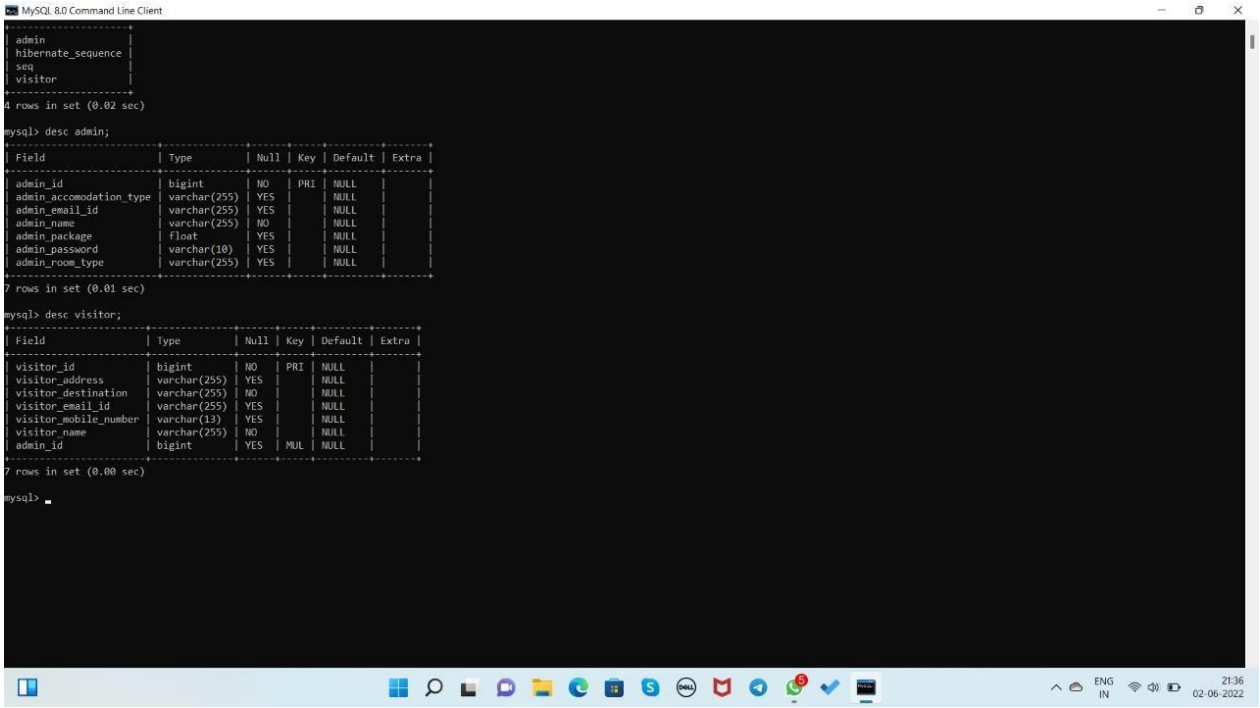
Database Table Design

Admin Table



SI NO	FIELD NAME	DATA TYPE
1	admin Id	Long
2	adminName	Varchar
3	admin Password	Varchar
4	adminEmailId	Varchar
5	adminAccomodationType	Varchar
6	adminPackage	Float
7	adminRoomType	Varchar

Visitor Table



SI NO	FIELD NAME	DATA TYPE
1	visitorId	Long
2	visitorName	Varchar
3	visitorMobileNumber	Varchar
4	visitorEmailId	Varchar
5	visitorAddress	Varchar
6	visitorDestination	Varchar

Conclusion:

Tourism Management System make visitors easy to choose tourist place and accommodation by giving affordable packages to them and easy place to find and sort visitor's information.