**A PROJECT REPORT ON**

# Hotel Management System

**Submitted by**

Ms. Kiranya Gaddam G

Ms. Nivetha K

Ms. Pavithra Dharmalingam D

Ms. Niranjana M

**Batch No. 6769**

**Under the Guidence of Trainer**

Mrs. Indrakka Mali

|  |  |
| --- | --- |
| **S. No** | **Topic Name** |
| **1** | **Introduction** |
| **2** | **Objective** |
| **3** | **Software Requirements** |
| **4** | **System overview and snapshots** |
| **5** | **Conclusion** |

**Introduction**

The project “**Hotel Management System**” is developed using spring boot framework, which mainly focuses on basic operations of customer room booking. Like getting the customer details to book the room, Insert customer Details, Delete customer Details and getting records of Customer Details based Account Id.

**Account Module:**

▪ Fetch all details of Customer records.

▪ Fetch customer record by Particular customer id.

▪ Fetch customer record by Particular customer room number.

▪ In this Module Customer can delete their room.

**Customer Module:**

▪ In the Booking Module:

▪ We Can Fetch all details of Customer records.

▪ In this module Customer Can Update Their Records like PhoneNo, password, etc.

▪ We can get the customer details and delete the record by using customer id.

**Objectives:**

▪The Main objective of study are

▪ To Understand the concept of Hotel room booking.

▪ To analyze the importance, function, advantages and limitation of customer booking.

▪ The Main objective is how customer provide efficient service to User.

▪ It provides “better and efficient” service”.

▪ Faster way to get information about the Customer and their details.

▪ Provide facility for proper monitoring and reduce paper work.

▪ All details will be available on a click.

**System Overview:**

▪ The Hotel Management System will be automated the traditional system.

▪ There is no need to use paper and pen.

▪ Checking a Customer and Their details is very easy.

▪ Adding new Customer record is very easy.

▪ Deleting or updating a record of a particular Customer Details is simple

**Requirements:**

**Software Requirement:**

▪ Database: MySQL

▪ API- Spring Data JPA, spring web, spring security

▪ Tools: Postman, IDE-Spring Tool Suit4

▪ Coding language -Java 1.8:

**Hardware Requirements:**

▪ RAM: 4GB

▪ Processor: 64bit

▪ Memory: 512 MB

▪ Disk Space: 100GB

▪ Windows:10

**Spring Tool Suite: -** 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

▪ MySQL is free

▪ MySQL is ideal for both small and large applications

▪ MySQL is very fast, reliable, scalable, and easy to use

▪ MySQL is cross-platform

**Program:**

**Controller:**

**Customer Controller:**

package com.example.demo.Controller;

/\*\*

\* In this customer controller Some annotations are used

\* @RestController->used for defining the class is controller

\* @Autowired->used for object injection

\* @GetMapping->used for select operation

\* @PutMapping->used for update operation

\* @DeleteMapping->used for delete operation

\*/

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.Customer;

import com.example.demo.error.CustomerNotFoundException;

import com.example.demo.service.CustomerService;

@RestController

public class CustomerController {

@Autowired

CustomerService customerService;

//save

//Gets all records

@GetMapping("/customers/")

public List<Customer> fetchCustomerList() {

return customerService.fetchCustomerList();

}

//get records by id

@GetMapping("/customers/{cid}")

public Customer fetchCustomerById(@PathVariable("cid")Long cid) throws CustomerNotFoundException

{

return customerService.fetchCustomerById(cid);

}

//delete Record by id

@DeleteMapping("/customers/{cid}")

public String deleteCustomerById(@PathVariable("cid")Long cid)

{

customerService.deleteCustomerById(cid);

return "customer id is deleted";

}

//update Record by id

@PutMapping("/customers/{cid}")

public Customer updateCustomerById(@PathVariable("cid")Long cid, @RequestBody Customer customer) throws CustomerNotFoundException

{

return customerService.updateCustomerById(cid,customer);

}

}

**Hotel Controller:**

package com.example.demo.Controller;

/\*\*

\* In this customer controller Some annotations are used

\* @RestController->used for defining the class is controller

\* @Autowired->used for object injection

\* @GetMapping->used for select operation

\* @PutMapping->used for update operation

\* @DeleteMapping->used for delete operation

\*/

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.Hotel;

import com.example.demo.error.HotelNotFoundException;

import com.example.demo.service.HotelService;

@RestController

public class HotelController {

@Autowired

HotelService hotelService;

//save

@PostMapping("/hotels/")

public Hotel saveHotel( @RequestBody Hotel hotel) {

return hotelService.saveHotel(hotel);

}

//Gets all records

@GetMapping("/hotels/")

public List<Hotel> fetchHotelList() {

return hotelService.fetchHotelList();

}

//get records by id

@GetMapping("/hotels/{id}")

public Hotel fetchHotelById(@PathVariable("id")Long hid) throws HotelNotFoundException

{

return hotelService.fetchHotelById(hid);

}

//get record by name

@GetMapping("/hotels/name/{name}")

public Hotel fetchHotelByName(@PathVariable("name")String hname)

{

return hotelService.fetchHotelByName(hname);

}

//get record by code

@GetMapping("/hotels/code/{code}")

public Hotel fetchHotelByCode(@PathVariable("code")String hcode)

{

return hotelService.fetchHotelByCode(hcode);

}

//get record by address

@GetMapping("/hotels/address/{address}")

public Hotel fetchHotelByAddress(@PathVariable("address")String haddress)

{

return hotelService.fetchHotelByAddress(haddress);

}

//get record by phone no

@GetMapping("/hotels/phno/{phno}")

public Hotel fetchHotelByPhno(@PathVariable("phno")int hphno)

{

return hotelService.fetchHotelByPhno(hphno);

}

//delete Record by id

@DeleteMapping("/hotels/{id}")

public String deleteHotelById(@PathVariable("id")Long hid) throws HotelNotFoundException

{

hotelService.deleteHotelById(hid);

return "Hotel is deleted";

}

//update Record by id

@PutMapping("/hotels/{id}")

public Hotel updateHotelById(@PathVariable("id")Long hid, @RequestBody Hotel hotel)throws HotelNotFoundException

{

return hotelService.updateHotelById(hid,hotel);

}

}

**Entity:**

**Customer:**

package com.example.demo.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.Range;

/\*\*

\* This customer class is converted as table and the variables are converted into columns

\* Here some annotations are used like entity,id,table,column,generated value...

\* Some of validations also performed not blank,not null,length

\*/

//This annotation is used to convert the class into the tables.

@Entity

@Table(name="customer\_table")

public class Customer {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private Long cid;

@NotBlank(message="Customer name cannot be blank")

private String cname;

@Range(message="Customer room cannot be less than 4 digits")

private int croomno;

@Column(length=10)

@NotBlank(message="Phno cannot be null")

private String cphno;

private String caddress;

public Customer() {

super();

}

public Customer(Long cid, @NotBlank(message = "Customer name cannot be blank") String cname,

@Range(message = "Customer room cannot be less than 4 digits") int croomno,

@NotBlank(message = "Phno cannot be null") String cphno, String caddress) {

super();

this.cid = cid;

this.cname = cname;

this.croomno = croomno;

this.cphno = cphno;

this.caddress = caddress;

}

public Long getCid() {

return cid;

}

public void setCid(Long cid) {

this.cid = cid;

}

public String getCname() {

return cname;

}

public void setCname(String cname) {

this.cname = cname;

}

public int getCroomno() {

return croomno;

}

public void setCroomno(int croomno) {

this.croomno = croomno;

}

public String getCphno() {

return cphno;

}

public void setCphno(String cphno) {

this.cphno = cphno;

}

public String getCaddress() {

return caddress;

}

public void setCaddress(String caddress) {

this.caddress = caddress;

}

@Override

public String toString() {

return "Customer [cid=" + cid + ", cname=" + cname + ", croomno=" + croomno + ", cphno=" + cphno + ", caddress="

+ caddress + "]";

}

}

**Error Message:**

package com.example.demo.entity;

import org.springframework.http.HttpStatus;

public class ErrorMessage {

private HttpStatus status;

private String Message;

public ErrorMessage() {

super();

}

public ErrorMessage(HttpStatus status, String message) {

super();

this.status = status;

Message = message;

}

public HttpStatus getStatus() {

return status;

}

public void setStatus(HttpStatus status) {

this.status = status;

}

public String getMessage() {

return Message;

}

public void setMessage(String message) {

Message = message;

}

@Override

public String toString() {

return "ErrorMessage [status=" + status + ", Message=" + Message + "]";

}

}

**Hotel:**

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.persistence.Table;

import javax.validation.constraints.NotBlank;

/\*\*

\* This customer class is converted as table and the variables are converted into columns

\* Here some annotations are used like entity,id,table,column,generated value...

\* Some of validations also performed not blank,not null,length

\*/

//This annotation is used to convert the class into the tables.

@Entity

@Table(name="hotel\_table")

public class Hotel {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

@SequenceGenerator(name="seq",initialValue=1000)

private Long hotelid;

@NotBlank(message="please add hotel name")

private String hotelName;

private String hotelCode;

private String hotelAddress;

private String hotelPhno;

@OneToMany(targetEntity=Customer.class,cascade=CascadeType.ALL)

@JoinColumn(name="customer\_id")

private List<Customer> customer;

public Hotel() {

super();

}

public Hotel(Long hotelid, @NotBlank(message = "please add hotel name") String hotelName, String hotelCode,

String hotelAddress, String hotelPhno, List<Customer> customer) {

super();

this.hotelid = hotelid;

this.hotelName = hotelName;

this.hotelCode = hotelCode;

this.hotelAddress = hotelAddress;

this.hotelPhno = hotelPhno;

this.customer = customer;

}

public Long getHotelid() {

return hotelid;

}

public void setHotelid(Long hotelid) {

this.hotelid = hotelid;

}

public String getHotelName() {

return hotelName;

}

public void setHotelName(String hotelName) {

this.hotelName = hotelName;

}

public String getHotelCode() {

return hotelCode;

}

public void setHotelCode(String hotelCode) {

this.hotelCode = hotelCode;

}

public String getHotelAddress() {

return hotelAddress;

}

public void setHotelAddress(String hotelAddress) {

this.hotelAddress = hotelAddress;

}

public String getHotelPhno() {

return hotelPhno;

}

public void setHotelPhno(String hotelPhno) {

this.hotelPhno = hotelPhno;

}

public List<Customer> getCustomer() {

return customer;

}

public void setCustomer(List<Customer> customer) {

this.customer = customer;

}

@Override

public String toString() {

return "Hotel [hotelid=" + hotelid + ", hotelName=" + hotelName + ", hotelCode=" + hotelCode + ", hotelAddress="

+ hotelAddress + ", hotelPhno=" + hotelPhno + ", customer=" + customer + "]";

}

}

**Error:**

**Customer Not Found Exception:**

package com.example.demo.error;

/\*\*

\* This customer not found exception class is

\* used to display the customer operation serror message in the output screen

\*

\*/

public class CustomerNotFoundException extends Exception {

private static final long serialVersionUID =1L;

public CustomerNotFoundException(String s) {

super(s);

}

}

**Hotel Not Found Exception:**

package com.example.demo.error;

/\*\*

\* This hotel not found exception class is

\* used to display the hotel operation serror message in the output screen

\*

\*/

public class HotelNotFoundException extends Exception {

private static final long serialVersionUID =1L;

public HotelNotFoundException(String s) {

super(s);

}

}

**Rest Response Entity Exception Handler:**

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 {

@ExceptionHandler(HotelNotFoundException.class)

//here the hotel exceptions are handling

public ResponseEntity<ErrorMessage> hotelNotFoundException(HotelNotFoundException exception,WebRequest request){

ErrorMessage message=new ErrorMessage(HttpStatus.NOT\_FOUND,exception.getMessage());

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(message);

}

//here the customer exceptions are handling

@ExceptionHandler(CustomerNotFoundException.class)

public ResponseEntity<ErrorMessage>CustomerNotFoundException(CustomerNotFoundException exception,WebRequest request) {

ErrorMessage message=new ErrorMessage(HttpStatus.NOT\_FOUND,exception.getMessage());//constructor

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body(message);

}

}

**Repository:**

**Customer Repository:**

package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.entity.Customer;

/\*\*

\* This customer not found exception class is

\* used to display the customer operation serror message in the output screen

\*

\*/

@Repository

public interface CustomerRepository extends JpaRepository<Customer,Long>{

**Hotel Repository:**

package com.example.demo.repository;

/\*\*

\* This customer not found exception class is

\* used to display the customer operation serror message in the output screen

\*

\*/

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.entity.Hotel;

public interface HotelRepository extends JpaRepository<Hotel,Long> {

Hotel getByHotelName(String hname);

Hotel getByHotelCode(String hcode);

Hotel getByHotelAddress(String haddress);

Hotel getByHotelPhno(int hphno);

//custom method

}

**Service:**

**Customer Service:**

package com.example.demo.service;

import java.util.List;

import javax.validation.Valid;

import com.example.demo.entity.Customer;

import com.example.demo.error.CustomerNotFoundException;

public interface CustomerService {

Customer saveCustomer(Customer customer);

//select customers

List<Customer> fetchCustomerList();

//select customers by id

Customer fetchCustomerById(Long cid) throws CustomerNotFoundException;

//delete customer by id

void deleteCustomerById(Long cid);

//update customer by id

Customer updateCustomerById(Long cid, Customer customer) throws CustomerNotFoundException;

**Customer Service Implements:**

package com.example.demo.service;

import java.util.List;

import java.util.Objects;

import java.util.Optional;

import javax.validation.Valid;

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

import org.springframework.stereotype.Service;

import com.example.demo.entity.Customer;

import com.example.demo.error.CustomerNotFoundException;

import com.example.demo.repository.CustomerRepository;

@Service

public class CustomerServiceImpl implements CustomerService {

@Autowired

CustomerRepository customerRepo;

@Override

//save the customers

public Customer saveCustomer(Customer customer) {

// TODO Auto-generated method stub

return customerRepo.save(customer);

}

@Override

//list all customers

public List<Customer> fetchCustomerList() {

// TODO Auto-generated method stub

return customerRepo.findAll();

}

@Override

//select customer by id

public Customer fetchCustomerById(Long cid) throws CustomerNotFoundException {

Optional<Customer> hotel=customerRepo.findById(cid);

if(!hotel.isPresent()) {

throw new CustomerNotFoundException("Customer Not Available");

}

// TODO Auto-generated method stub

return customerRepo.findById(cid).get();

}

@Override

//delete customer by id

public void deleteCustomerById(Long cid) {

// TODO Auto-generated method stub

customerRepo.deleteById(cid);

}

@SuppressWarnings("unlikely-arg-type")

@Override

//update customer by id

public Customer updateCustomerById(Long cid, Customer customer) throws CustomerNotFoundException {

// TODO Auto-generated method stub

Optional<Customer> hotel1=customerRepo.findById(cid);

Customer customerDB=null;

if(hotel1.isPresent()) {

customerDB=customerRepo.findById(cid).get();

if(Objects.nonNull(customer.getCname()) && !"".equalsIgnoreCase(customer.getCname())) {

customerDB.setCname(customer.getCname());

}

if(Objects.nonNull(customer.getCaddress()) && !"".equalsIgnoreCase(customer.getCaddress())) {

customerDB.setCaddress(customer.getCaddress());

System.out.println(customer.getCaddress());

}

if(Objects.nonNull(customer.getCphno()) && !"".equals(customer.getCphno())) {

customerDB.setCphno(customer.getCphno());

System.out.println(customer.getCaddress());

}

if(Objects.nonNull(customer.getCroomno()) && !"".equalsIgnoreCase(customer.getCaddress())) {

customerDB.setCroomno(customer.getCroomno());

System.out.println(customer.getCaddress());

}

return customerRepo.save(customerDB);

}

else {

throw new CustomerNotFoundException("Customer Not Available");

}

}

}

**Hotel Service:**

package com.example.demo.service;

import java.util.List;

import javax.validation.Valid;

import com.example.demo.entity.Hotel;

import com.example.demo.error.HotelNotFoundException;

public interface HotelService {

public Hotel saveHotel(@Valid Hotel hotel);

public List<Hotel> fetchHotelList();

public Hotel fetchHotelById(Long hid) throws HotelNotFoundException;

//

public void deleteHotelById(Long hid) throws HotelNotFoundException;

public Hotel updateHotelById(Long hid, Hotel hotel) throws HotelNotFoundException;

public Hotel fetchHotelByName(String hname);

public Hotel fetchHotelByCode(String hcode);

public Hotel fetchHotelByAddress(String haddress);

public Hotel fetchHotelByPhno(int hphno);

}

**Hotel Service Implement:**

package com.example.demo.service;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

import java.util.Optional;

import javax.validation.Valid;

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

import org.springframework.stereotype.Service;

import com.example.demo.entity.Customer;

import com.example.demo.entity.Hotel;

import com.example.demo.error.CustomerNotFoundException;

import com.example.demo.error.HotelNotFoundException;

import com.example.demo.repository.HotelRepository;

@Service

public class HotelServiceImpl implements HotelService{

@Autowired

HotelRepository hotelRepo;

@Override

public Hotel saveHotel(@Valid Hotel hotel) {

// TODO Auto-generated method stub

return hotelRepo.save(hotel);

}

@Override

public List<Hotel> fetchHotelList() {

// TODO Auto-generated method stub

return hotelRepo.findAll();

}

@Override

public Hotel fetchHotelById(Long hid) throws HotelNotFoundException {

// TODO Auto-generated method stub

Optional<Hotel> hotel=hotelRepo.findById(hid);

if(!hotel.isPresent()) {

throw new HotelNotFoundException("Hotel Not Available");

}

// TODO Auto-generated method stub

return hotelRepo.findById(hid).get();

}

@Override

public void deleteHotelById(Long hid) {

// TODO Auto-generated method stub

hotelRepo.deleteById(hid);

}

@Override

public Hotel updateHotelById(Long hid, Hotel hotel) throws HotelNotFoundException {

// TODO Auto-generated method stub

Optional<Hotel> hotel1=hotelRepo.findById(hid);

Hotel hotDB=null;

if(hotel1.isPresent()) {

hotDB=hotelRepo.findById(hid).get();

if(Objects.nonNull(hotel.getHotelName()) && !"".equalsIgnoreCase(hotel.getHotelName())) {

hotDB.setHotelName(hotel.getHotelName());

}

if(Objects.nonNull(hotel.getHotelAddress()) && !"".equalsIgnoreCase(hotel.getHotelAddress())) {

hotDB.setHotelAddress(hotel.getHotelAddress());

System.out.println(hotel.getHotelAddress());

}

if(Objects.nonNull(hotel.getHotelCode()) && !"".equalsIgnoreCase(hotel.getHotelCode())) {

hotDB.setHotelCode(hotel.getHotelCode());

System.out.println(hotel.getHotelCode());

}

return hotelRepo.save(hotDB);

}

else {

throw new HotelNotFoundException("Hotel Not Available");

}

}

@Override

public Hotel fetchHotelByName(String hname) {

// TODO Auto-generated method stub

return hotelRepo.getByHotelName(hname);

}

@Override

public Hotel fetchHotelByCode(String hcode) {

// TODO Auto-generated method stub

return hotelRepo.getByHotelCode(hcode);

}

@Override

public Hotel fetchHotelByAddress(String haddress) {

// TODO Auto-generated method stub

return hotelRepo.getByHotelAddress(haddress);

}

@Override

public Hotel fetchHotelByPhno(int hphno) {

// TODO Auto-generated method stub

return hotelRepo.getByHotelPhno(hphno);

}

}

**Hotel Spring JPA Project Application:**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HotelSpringJpaProjectApplication {

public static void main(String[] args) {

SpringApplication.run(HotelSpringJpaProjectApplication.class, args);

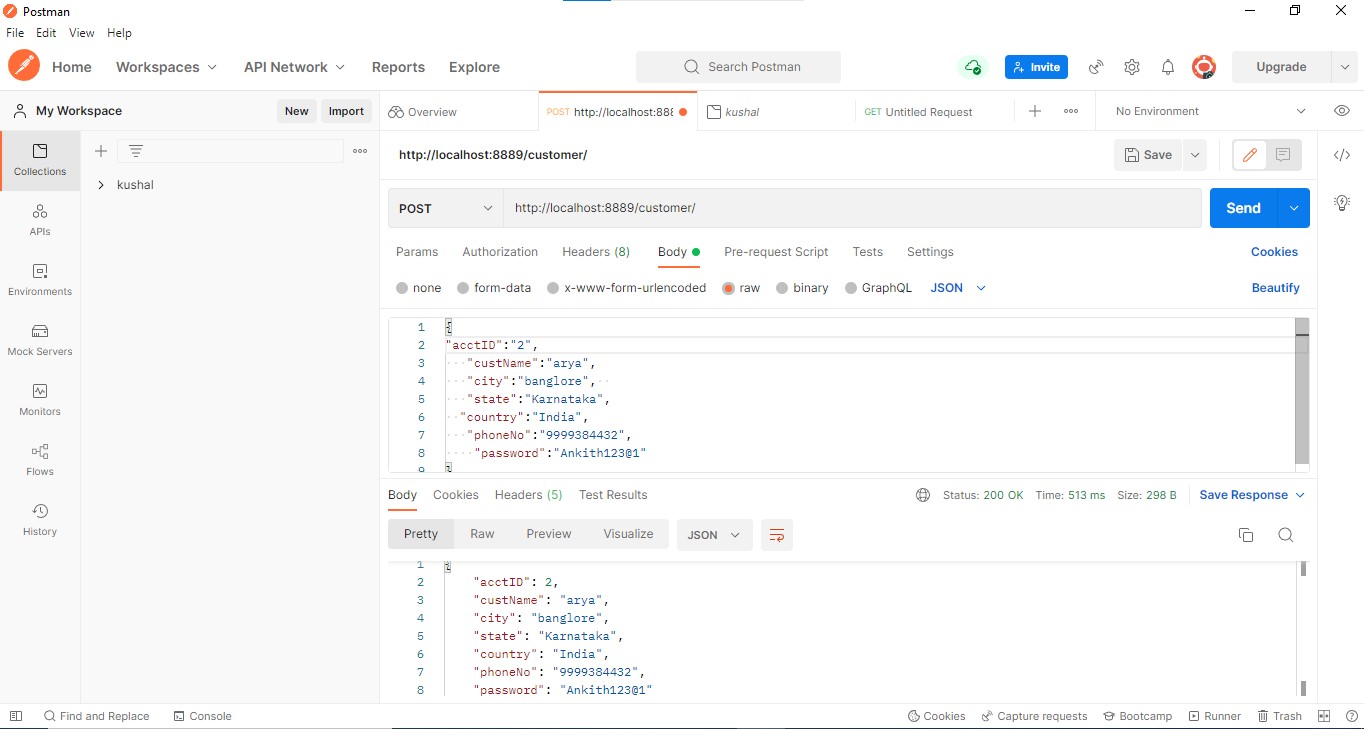
}

}

**Screenshot**

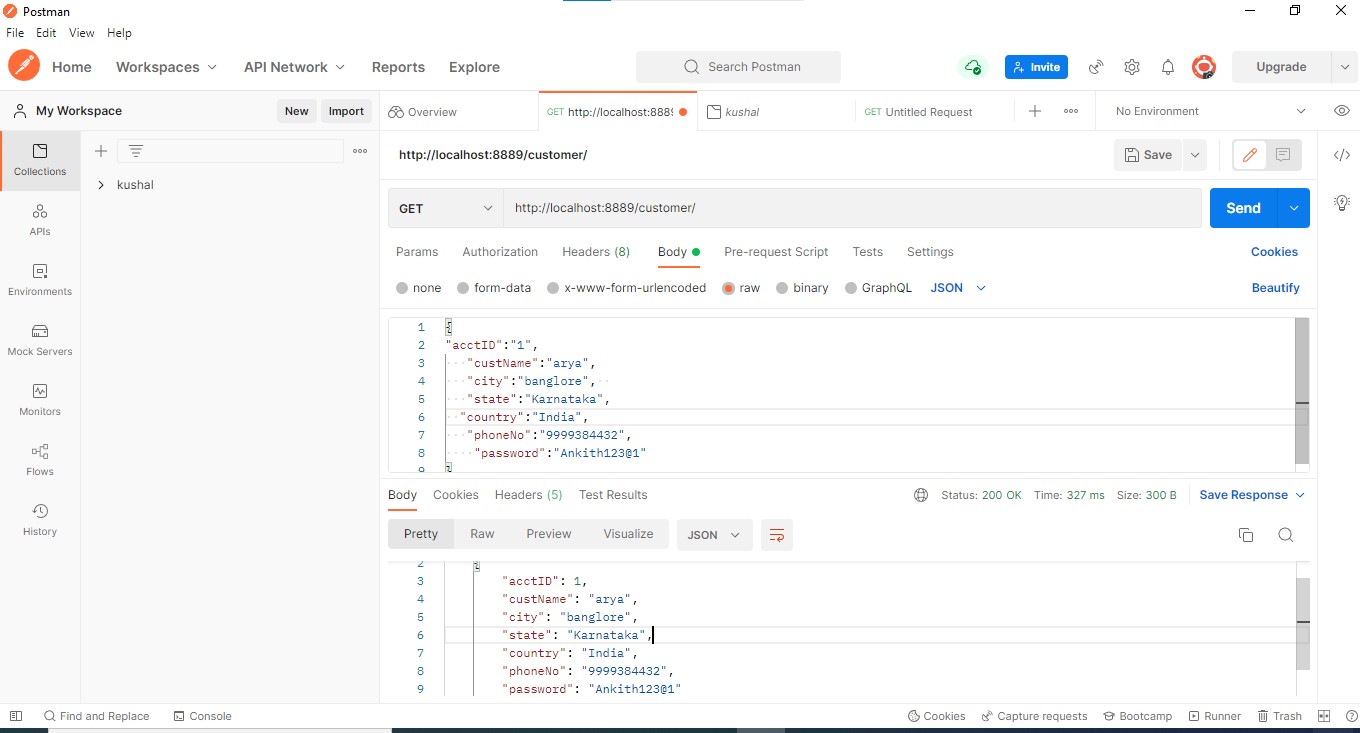
Step 1: we can Inserting the customer Details using post method.

URL: <http://localhost:8809/customer/>



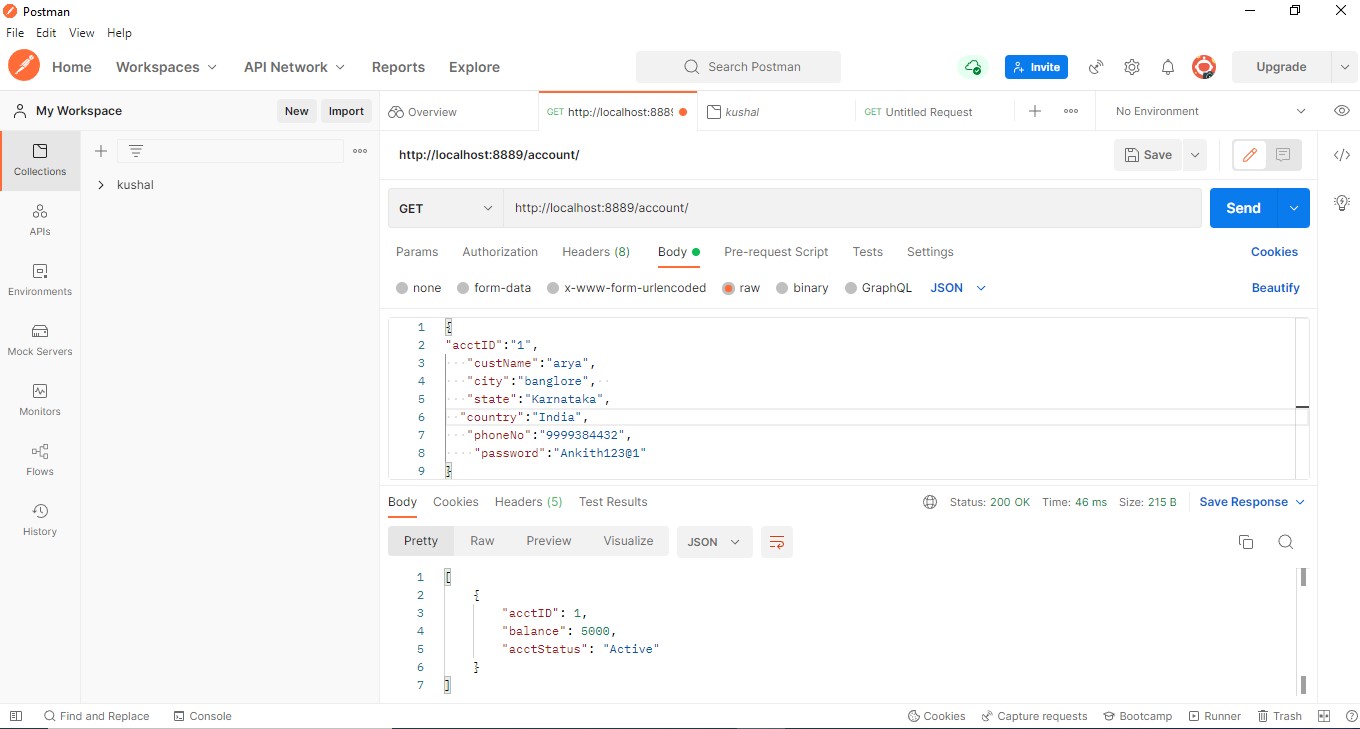
Step 2: Display All details of Customer Using Get Method.

URL:<http://localhost:8809/customer/>



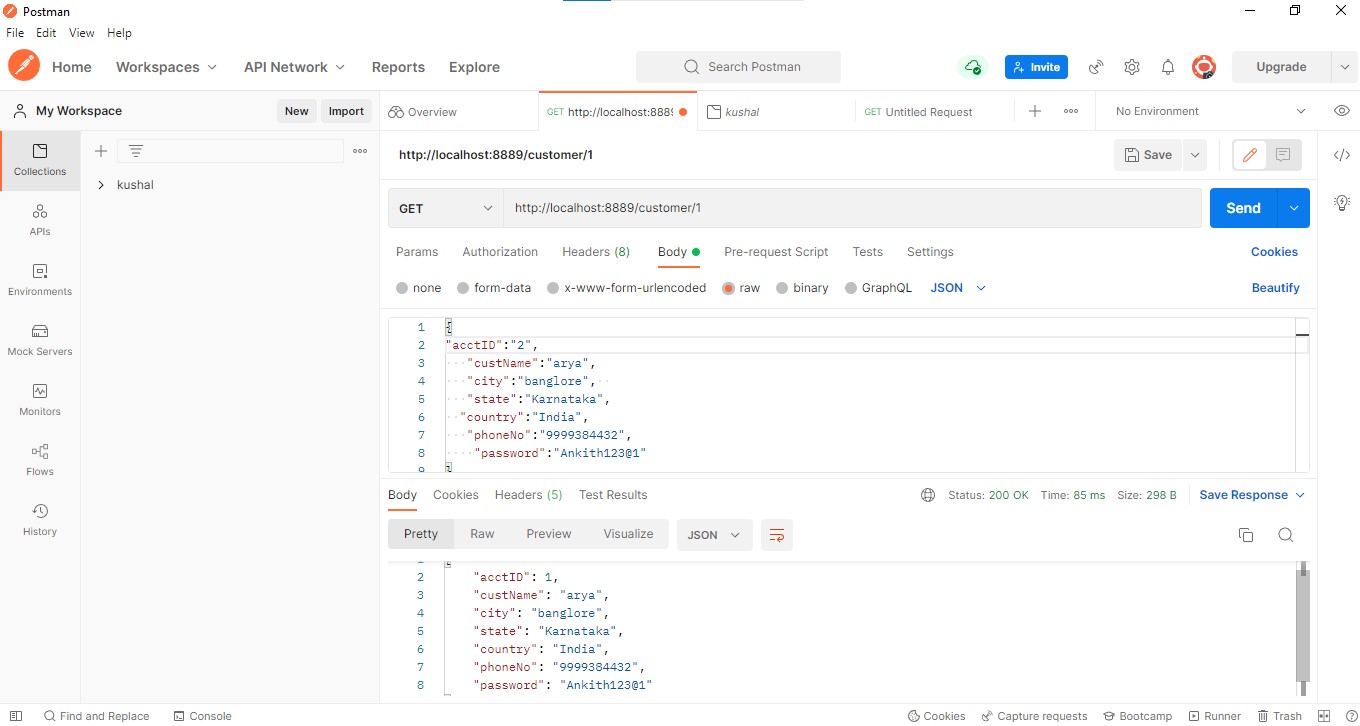
Step 3: Display All details of Account.

URL:<http://localhost:8809/Hotal/>

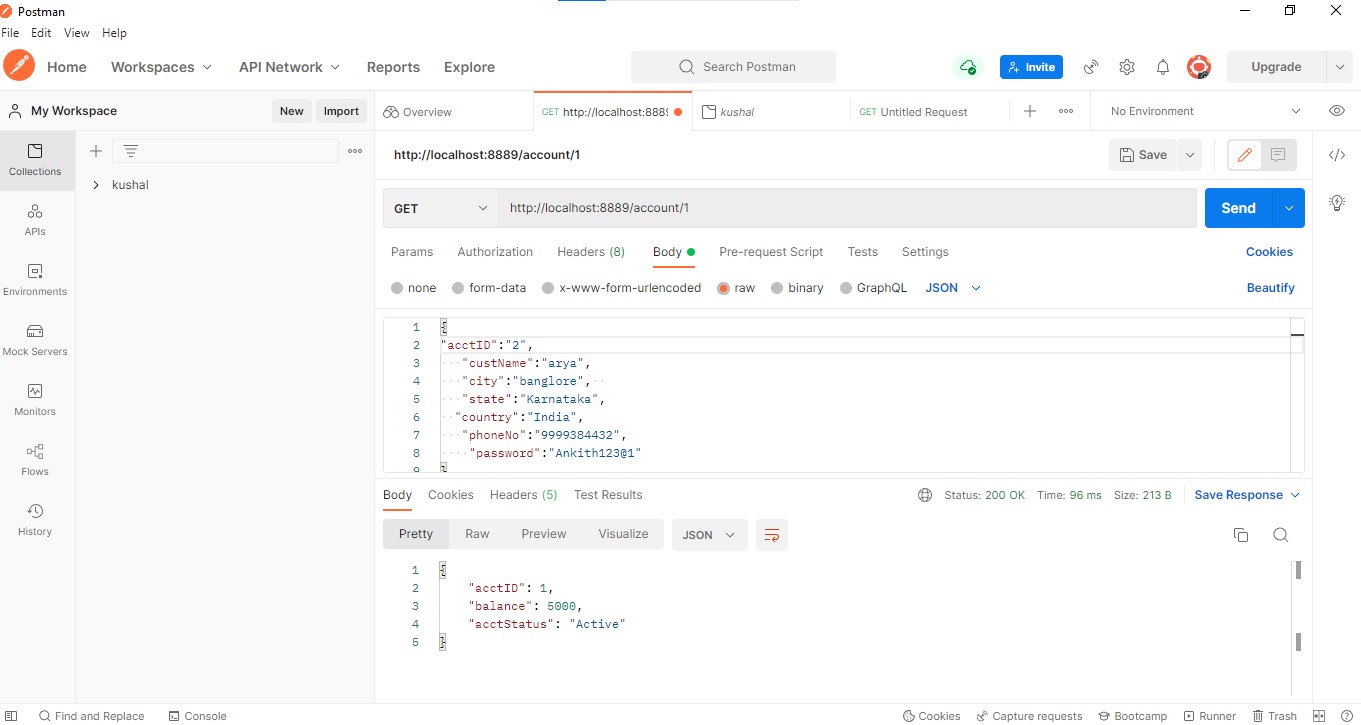


Step 4: Get Particular details of customer based on id.

URL:<http://localhost:8809/customer/>[1](http://localhost:8889/customer/)

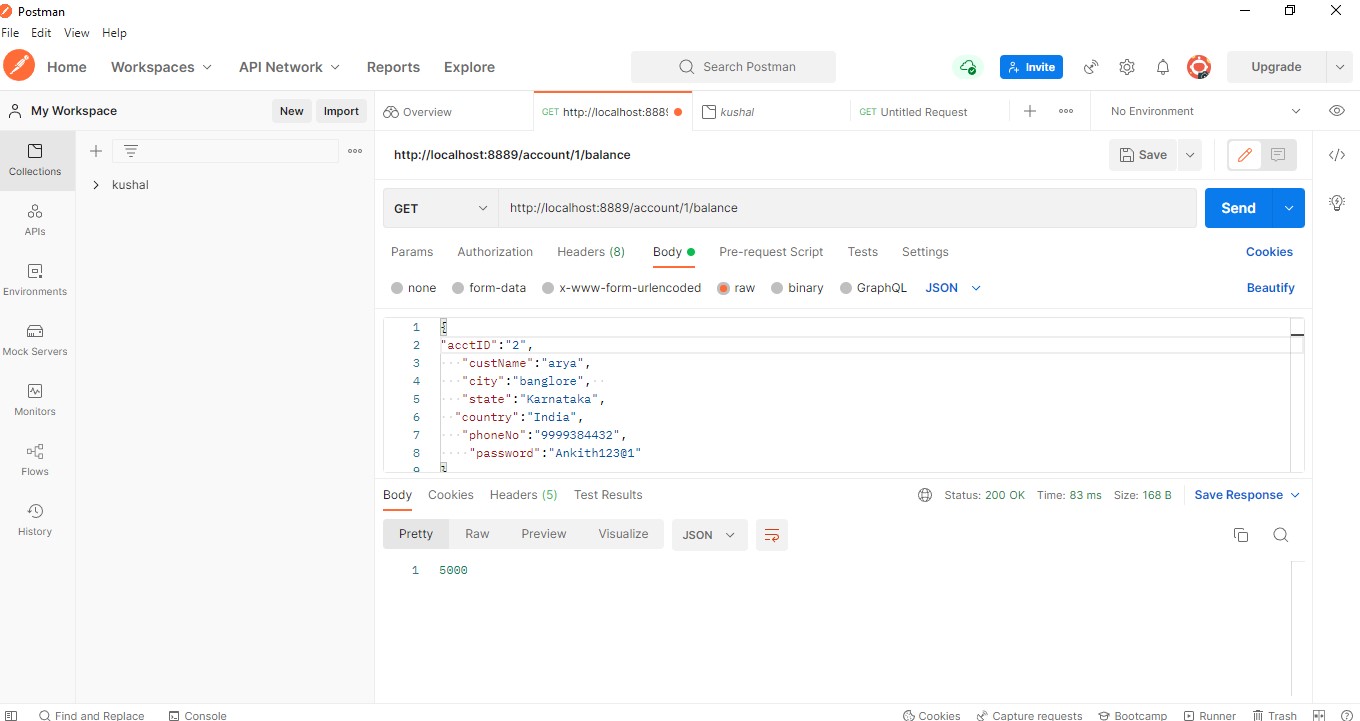


Step 5:Get Particular details of customer Details based on id Using Get Method. URL:<http://localhost:8809/Hotel/1>



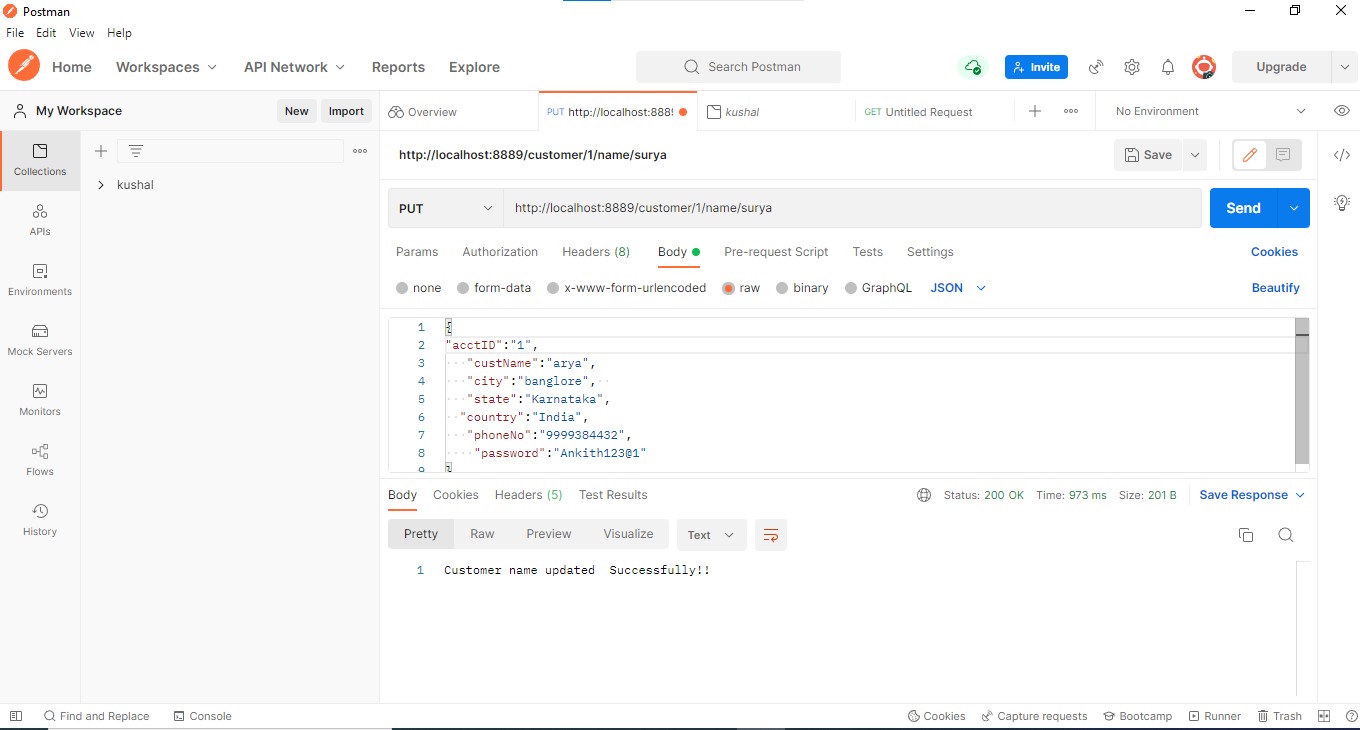
Step 6: GET the balance of customer based on id

URL :http://localhost:8809/Customer/1/



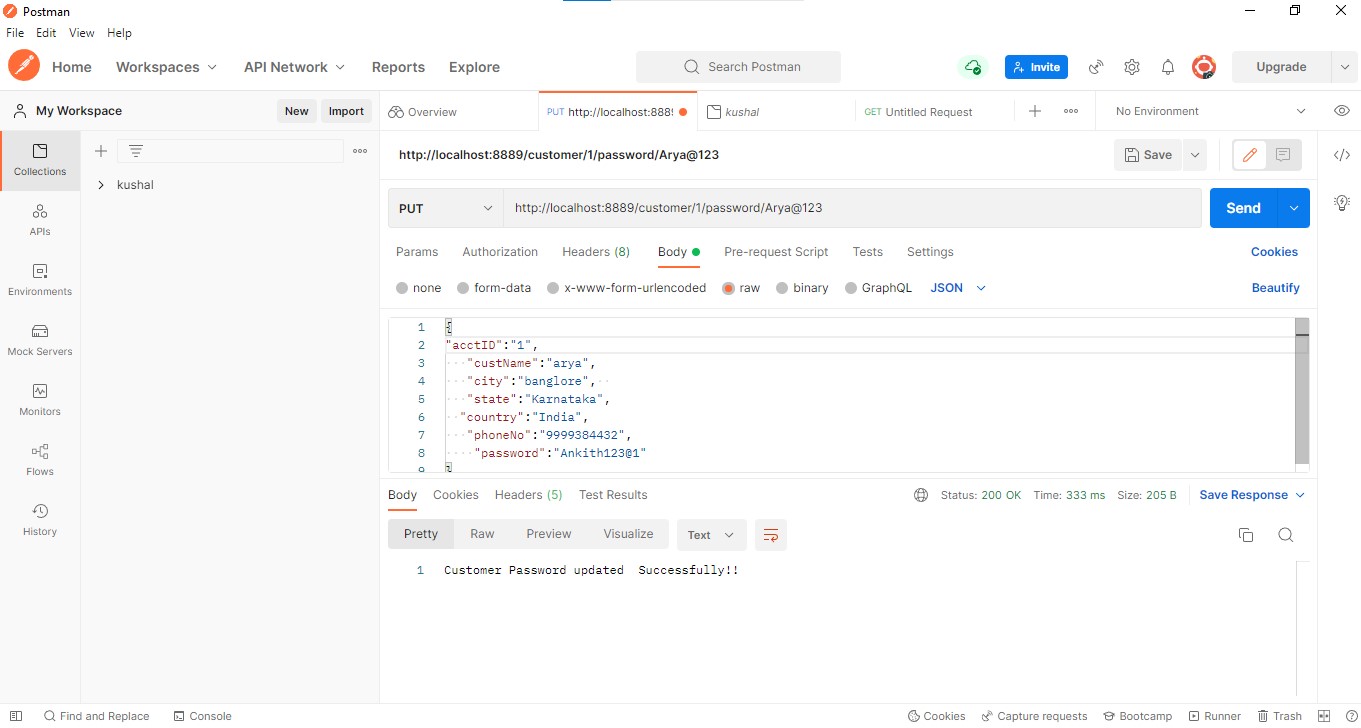
Step 7: If Customer wants to Change His Details like Name .We can update customer Details Based on Id using PUT Method.

URL: http://localhost:8809/customer/2/name/Surya



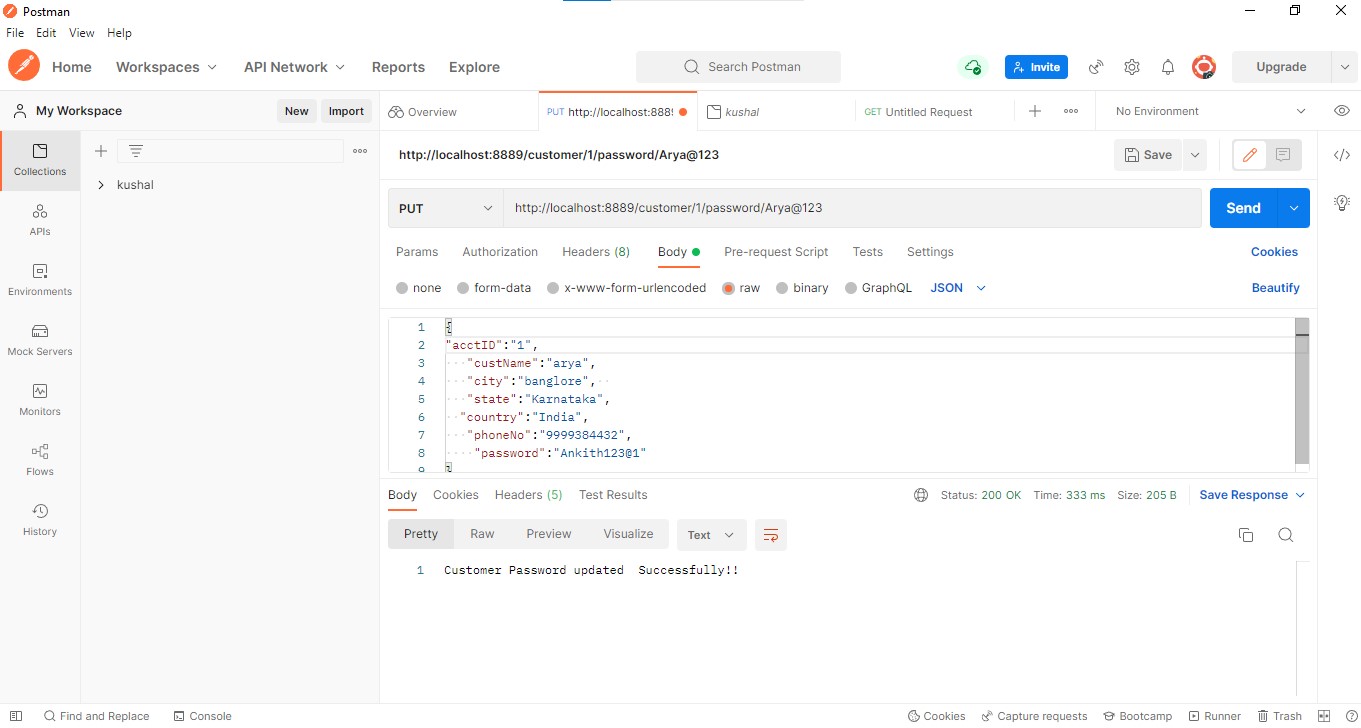
Step 8: IF Customer wants to Change His City based on id Using PUT Method.

URL: <http://localhost:8809/customer/4/city/Bangalore>



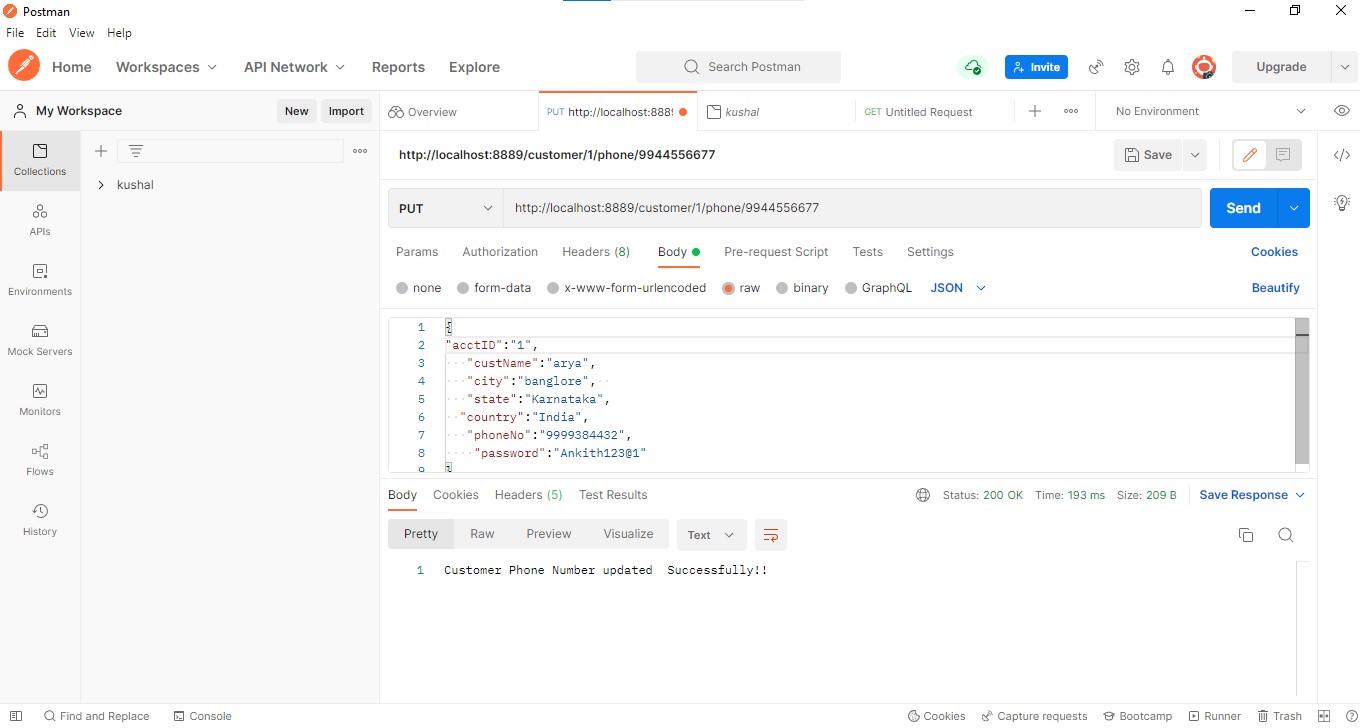
Step 9: IF Customer wants to Change His Password Details.

URL:<http://localhost:8809/customer/4/password/prash123>

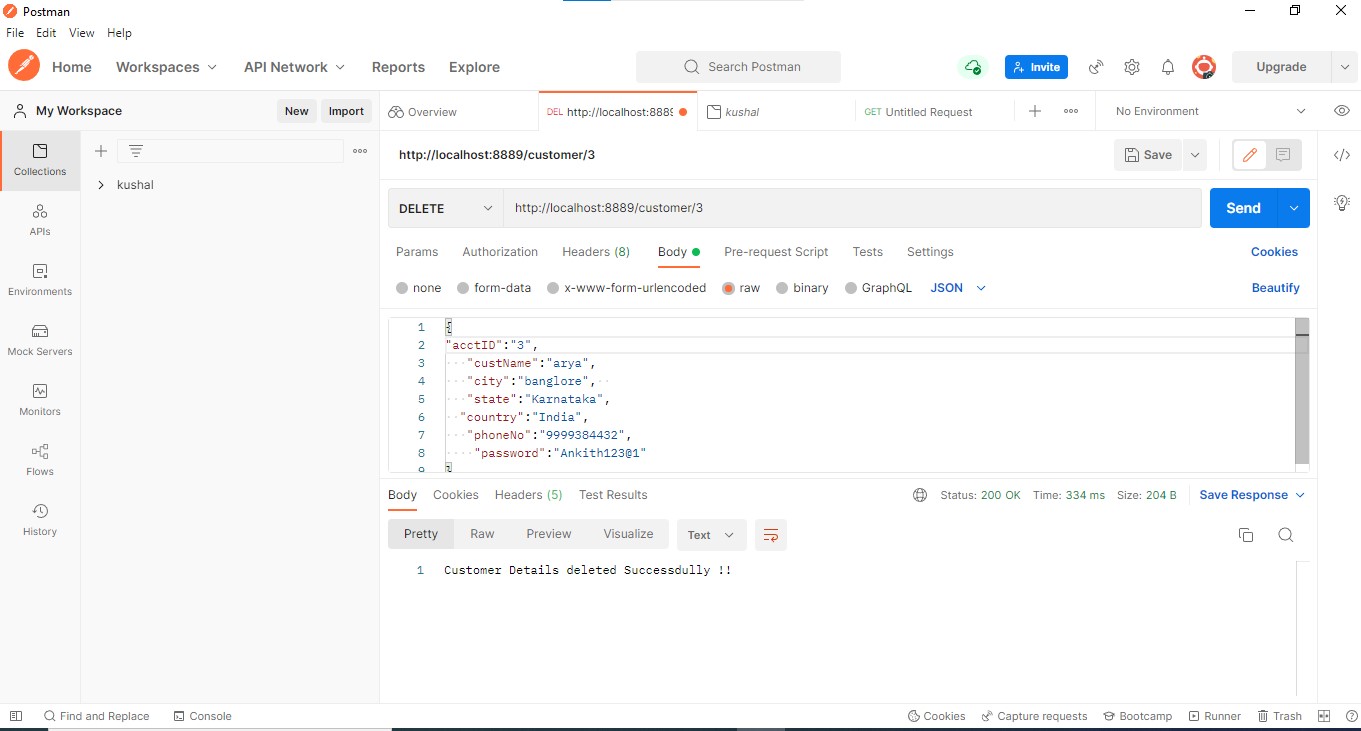


Step 10: If Customer wants to Change His Phone Number details.

URL[:http://localhost:8809/customer//phone/9980765432](http://localhost:8889/customer/4/phone/9980765432)

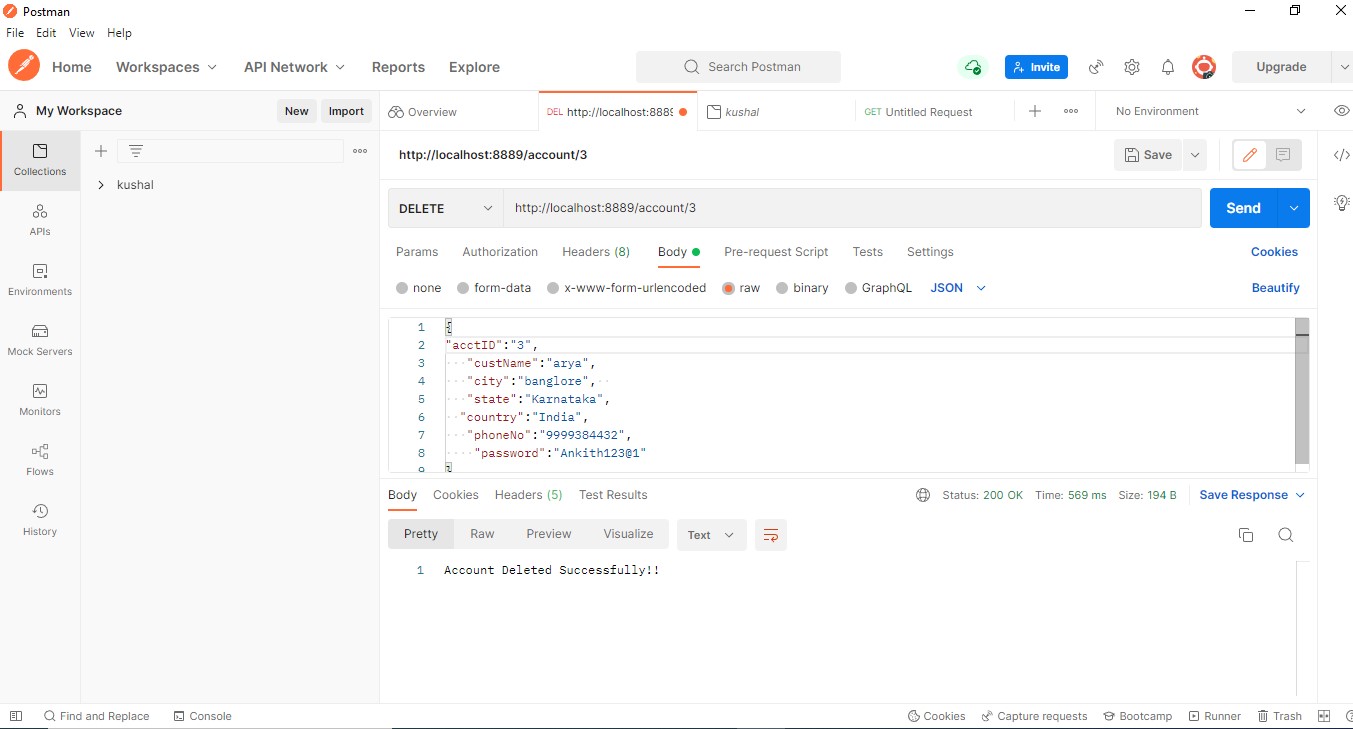


Step 11: Hotel can delete their customer details based on id Using Delete Method. URL:<http://localhost:8809/customer/2>



Step 12:Hotel delete their account details based on id Using Delete Method.

URL: http://localhost:8809/Hotel/1



**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.

1. **@Repository:**

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

1. **@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 autowire relationships between collaborating beans. This is called spring bean autowiring.

1. **@GetMapping**:

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

1. **@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.

1. **@Configuration:**

Spring @Configuration annotation helps in spring annotation based configuration. @Configuration annotation indicates that a class declares one or more @Bean methods and may be processed by the spring container to generate bean definitions and service requests for those beans at runtime

1. **@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.

1. **@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

1. **@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.

1. **@ Exception Handler:**

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

1. **@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.

1. **@BeanAnnotation:**

Spring @Bean annotation tells that a method produces a bean to be managed by the spring container. It is a method-level annotation. During Java configuration (@Configuration), the method is executed and its return value is registered as a bean within a BeanFactory.

**13.@Modifying:**

The @Modifying annotation is used to enhance the @query annotation so that we can execute not only select queries,but also Insert,Update,Delete,and even DDl queries.

## 14.@Query

The @Query annotation can only be used to annotate repository interface methods. The call of the annotated methods will trigger the execution of the statement found in it, and their usage is pretty straightforward. The @Query annotation supports both native SQL and JPQL.

**15.@Delete Mapping.**

@DeleteMapping is a specialized version that acts as a shortcut for Request Mapping The DELETE HTTP method is used to delete the resource and @DeleteMapping annotation maps the HTTP DELETE requests onto specific handler methods of a Spring controller. You can use this annotation only at the method level. You can use only the @RequestMapping annotation at the class level.

**16.@Rest Controller.**

@RestController is used for making restful web services with the help of the @RestController annotation.This annotation is used at the class level and allows the class to handle the requests. The RestController allows to handle all REST APIs such as GET, POST, Delete, PUT requests.

**Database Table Design:**

**Hotel Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Field Name** | | **Data Type** |
| **1** | **Name** | | **String** |
| **2** | **Code** | | **Int** |
| **3** | **Address** | | **Sting** |
| **4** | | **Phone number** | **Long** |

**Customer Table:**

|  |  |  |
| --- | --- | --- |
| **SL NO** | **Field Name** | **Data Type** |
| **1** | **Name** | **String** |
| **2** | **Room Number** | **Int** |
| **3** | **Phone Number** | **Int** |
| **4** | **Address** | **String** |

**Conclusion:**

Hotel Management System make online booking Service more accessible t o customer

By giving them an Easy palce to find and sort Information like get their room

Inforamtion details,Online ,modify their details.

Internet booking is changing the booking industry and is having the major effects on hotels relationships. “now is more of a norm rather than an exception in many developed countries" due to the fact that it is the economical way of providing hotel room booking services.

In a country like India, there is need for providing better and customized services to the customers. Hotels must be concerned about the attitudes of customers with regard to acceptance of internet booking. The importance of security and privacy for acceptance it was found that people claim that they have knowledge about security issues but they have no clear idea about all kinds of frauds. The present study shows that customers are more reluctant to accept new technologies or methods that might contain little risk. Hence, hotels should design the website to address security and trust issues. The basic objective of my research was to analyze the awareness among customers for internet room booking in India. It gives direction to research tools, research types and techniques.