A

Project

Report on

HOSPITAL MANAGEMENT SYSTEM

Submitted To

Batch No.2021-6769

Under the Guidence of

Trainer Mrs. Indrakka Mali

Submitted By

Mrs. M. Dharani

Ms. B. Keshava Devi

Ms. K. Priyadharshini

Ms. M. Srimugi

Ms. M. Tharani

INDEX

Introduction:

- ✓ The project "Hospital Management System" is developed using spring boot framework, which mainly focuses on basic operations. Like Inserting, Deleting, Updating and getting all records of Doctor and Patient information.
- ✓ It is accessible by Receptionist. Only they can add and delete the data into the database .The data can be retrieved easily. The data are well protected for personal use and make the data processing very fast.
- ✓ The project Hospital Management System included the registration of Patient, Storing the details into the system by using the database. The software has the facility to give a unique ID for every Patients and Doctors and stores the details of every patient and staff manually.

Receptionist Module:

- ✓ Inserting the Patients details.
- ✓ Fetch all Patients records.
- ✓ Fetch Patient details by Id.
- ✓ Fetch Patient details by name.
- ✓ Deleted the Patients details by Id.
- ✓ Update the Patients details by Id.

Doctor Module:

- ✓ Fetch Doctor Details by Id.
- ✓ Fetch all Doctor Records.
- ✓ Inserting the Doctor details.
- ✓ Fetch Doctor Details by name.
- ✓ Deleted the Doctor details by Id.
- ✓ Update the Doctor details by Id.

Patient Module:

- ✓ Fetch Patient details by Id.

Objectives:

- ✓ It provides “better and efficient” service”.
- ✓ Faster way to get information about the Patients.
- ✓ Provide facility for proper monitoring and reduce paper work.
- ✓ All details will be available on a click.

System Overview:

- ✓ The Hospital Management System will be automated the traditional system.
- ✓ There is no need to use paper and pen.
- ✓ Checking a Appointment record is very easy.
- ✓ Adding new Patient record is very easy.
- ✓ Deleting or updating a record of a particular Patient and Doctor is simple.

Software Requirement

Content	Description
Language	Java 1.8
Database	MYSQL
Framework	Hibernate, Spring Boot
API	Spring Data JPA, Spring Web, Validation
Tools	Postman, IDE Spring Tool Suit
Dependency Manager	Maven
Server	Apache Tomcat

Hardware Requirement

Content	Description
Processors	1.40GHz
RAM	8GB
Operating System	Window 10

Spring Tool Suit

Spring Tool Suit (STS) is a java IDE tailored for developing Spring-based enterprise applications. It is easier, faster, and more convenient. And most importantly it is based on Eclipse IDE. STS is free, open-source. Spring Tools 4 is the next generation of Spring tooling for the favorite coding environment. Largely rebuilt from scratch, it provides world-class support for developing Spring-based enterprise applications, whether you prefer Eclipse, Visual Studio Code, or Theia IDE.

Postman

Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated.

Method

Postman offers many endpoint interaction methods. The following are some of the most used, including their functions:

- ✓ GET: Obtain information
- ✓ POST: Add information
- ✓ PUT: Update certain information
- ✓ DELETE: Delete information

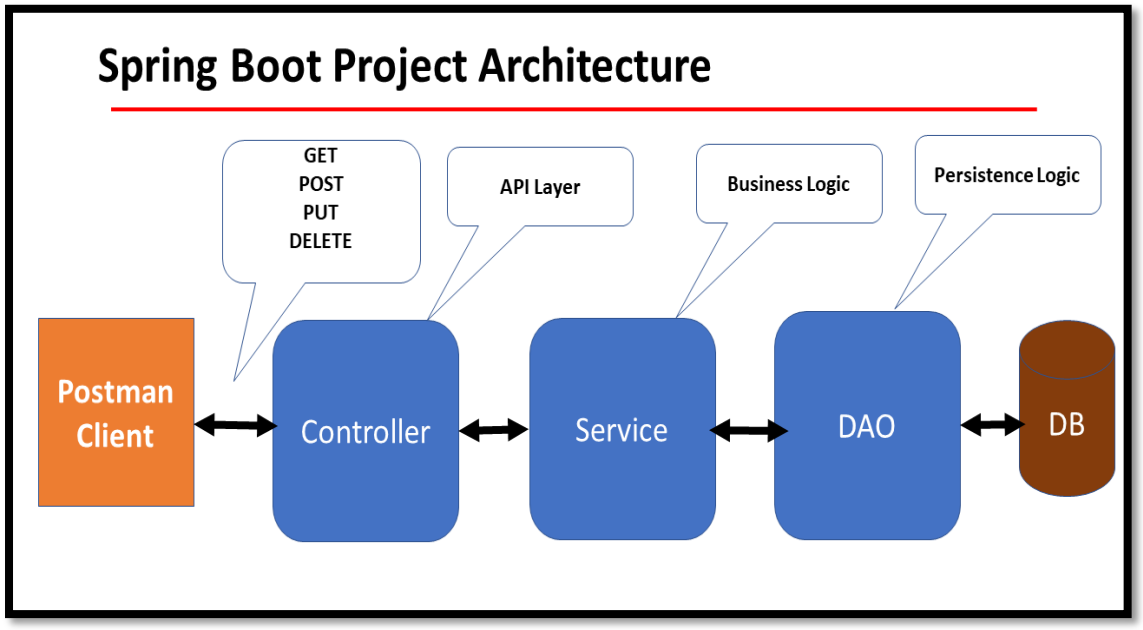
MySQL:

- ✓ MySQL is a Relational Database Management System(RDBMS) developed by Oracel that is based on Structured Query Language(SQL).
- ✓ A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or a place to hold the vast amounts of information in a corporate network. In particular, a relational database is a digital store collecting data and organizing it according to the relational model. In this model, tables consist of rows and columns, and relationships between data elements all follow a strict logical structure. An RDBMS is simply the set of software tools used to actually implement, manage, and query such a database.

Spring Boot:

Spring Boot is an open source, microservice-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase.

Architecture:



Spring Data JPA

Spring Boot JPA is a Java specification for managing relational data in Java applications. It allows us to access and persist data between Java object/ class and relational database.

CRUD Operation

- ✓ The CRUD stands for Create, Read/Retrieve, Update, and Delete. These are the four basic functions of the persistence storage.
- ✓ The CRUD operation can be defined as user interface conventions that allow view, search, and modify information through computer-based forms and reports. CRUD is data-oriented and the standardized use of HTTP action verbs. HTTP has a few important verbs.

  POST: Creates a new resource

  GET: Reads a resource

  PUT: Updates an existing resource

  DELETE: Deletes a resource
- ✓ Within a database, each of these operations maps directly to a series of commands. However, their relationship with a RESTful API is slightly more complex.

Standard CRUD Operation

- ✓ ***CREATE Operation***: It performs the INSERT statement to create a new record.
- ✓ ***READ Operation***: It reads table records based on the input parameter.
- ✓ ***UPDATE Operation***: It executes an update statement on the table. It is based on the input parameter.
- ✓ ***DELETE Operation***: It deletes a specified row in the table. It is also based on.

CRUD Operations Works

- ✓ CRUD operations are at the foundation of the most dynamic websites. Therefore, we should differentiate CRUD from the HTTP action verbs.

- ✓ Suppose, if we want to create a new record, we should use HTTP action verb POST. To update a record, we should use the PUT verb. Similarly, if we want to delete a record, we should use the DELETE verb. Through CRUD operations, users and administrators have the right to retrieve, create, edit, and delete records online.
- ✓ We have many options for executing CRUD operations. One of the most efficient choices is to create a set of stored procedures in SQL to execute operations.
- ✓ The CRUD operations refer to all major functions that are implemented in relational database applications. Each letter of the CRUD can map to a SQL statement and HTTP methods.

Operation	**SQL**	**HTTP Verbs**	**RESTful Web Service**
Create	Insert	Put/Post	Post
Read	Select	Get	Get
Update	Update	Put/Post/Patch	Put
Delete	Delete	Delete	Delete

Annotations:

- ✓ **@Springbootapplication**

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

- ✓ **@Entity**

  It specifies that the class is an entity and is mapped to a database table.

- ✓ **@Id**

  It is specifies the primary key of the object's table.

- ✓ **@RestController**

  It is a convenience annotation for creating Restful controllers. It is a specialization of @Component and is auto detected through class path scanning. It adds the @Controller and @ResponseBody annotations. It converts the response to JSON.

- ✓ **@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.

- ✓ **@Repository**

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

- ✓ **@Autowired**

  It is used for dependency injection. In spring boot application, all loaded beans are eligible for auto wiring to another bean.

- ✓ **@GeneratedValue**

  If we want to automatically generate the primary key value, we can add the @GeneratedValue annotation. This can use four generation types: auto,

identity, sequence and table. If we don't explicitly specify a value, the generation type defaults to auto.

- ✓ **@Table**

  It is used for adding the table name in the particular MySQL database.

- ✓ **@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.

- ✓ **@Column**

  It is used for adding the column the name in the table of a particular MySQL database.

- ✓ **@NotNull**

  @notnull annotation is a method should not return null. Variable cannot hold a null value.

- ✓ **@NotBlank**

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

- ✓ **@Email**

  It is a most common use case to have Email_id 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.

- ✓ **@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.

- ✓ **@Length**

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

- ✓ **@Range**

  @Range attribute for validating a particular field we dynamically change the values of min and max instead of hard coding.

- ✓ **@GetMapping**

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

- ✓ **@PutMapping**

  The PUT HTTP method is used to update the resource and @PutMapping annotation for mapping HTTP PUT requests onto specific handler methods. Specifically, @PutMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PUT).

- ✓ **@PostMapping**

  The POST HTTP method is used to create a resource and annotation for mapping HTTP POST requests onto specific handler methods.

Specifically, @PostMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST).

- ✓ **@DeleteMapping**

The DELETE HTTP method is used to delete the resource and @DeleteMapping annotation for mapping HTTP DELETE requests onto specific handler methods. Specifically, @DeleteMapping is a composed annotation that acts as a shortcut @RequestMapping(method =RequestMethod.DELETE).

- ✓ **@RequestBody**

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

- ✓ **@PathVariable**

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

- ✓ **@ExceptionHandler**

It allows us to handle specified types of exceptions through one single method.

- ✓ **@ResponseStatus**

@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.

- ✓ **@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.

Coding

Demo Package

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HospitalManagementProject1Application {

	public static void main(String[] args) {
	SpringApplication.run(HospitalManagementProject1Application.class, args);
}
}
```

Pojo Package

Class Doctor

```
package com.example.demo.entity;
import java.util.List;
```

```
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import org.hibernate.validator.constraints.Length;

@Entity
@Table(name = "Doctor_Details")
public class Doctor {
	@Id
	@NotNull(message = "Doctor id cannot be Null")
	private int did;
	@Column(name = "DoctorName")
	private String name;
	@Column(name = "DoctorMail",unique=true)
	@Email
	private String emailid;
	@Column(name = "MobileNumber")
	@Length(min = 10, max = 13, message = "Mobile number cannot be less than 10
Character")
	private String contactno;
	@Column(name = "Designation")
	@NotBlank(message = "Please fill the Designation field")
```

```
	private String designation;
	@Column(name = "Speciality")
	@NotBlank(message = "Speciality field cannot be Empty")
	private String speciality;
	@Column(name = "VisitingHours")
	private String visitinghours;
	@OneToMany(targetEntity = Patient.class,cascade = CascadeType.ALL)
	@JoinColumn(name="doctor_id")
	private List<Patient> patient;

	public Doctor() {
		super();
	}

	public Doctor(int did, String name, @Email String emailid,
			@Length(min = 10, max = 13, message = "Mobile number cannot be less than 10 Character") String contactno,
			@NotBlank(message = "Please fill the Designation field") String designation,
			@NotBlank(message = "Speciality field is empty") String speciality, String visitinghours,
			List<Patient> patient) {
		super();
		this.did = did;
		this.name = name;
		this.emailid = emailid;
		this.contactno = contactno;
		this.designation = designation;
		this.speciality = speciality;
```

```
        this.visitinghours = visitinghours;
        this.patient = patient;
}
public int getDid() {
        return did;
}
public void setDid(int did) {
        this.did = did;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public String getEmailid() {
        return emailid;
}
public void setEmailid(String emailid) {
        this.emailid = emailid;
}
public String getContactno() {
        return contactno;
}
public void setContactno(String contactno) {
        this.contactno = contactno;
}
public String getDesignation() {
        return designation;
```

```
	}
	public void setDesignation(String designation) {
		this.designation = designation;
	}
	public String getSpeciality() {
		return speciality;
	}
	public void setSpeciality(String speciality) {
		this.speciality = speciality;
	}
	public String getVisitinghours() {
		return visitinghours;
	}
	public void setVisitinghours(String visitinghours) {
		this.visitinghours = visitinghours;
	}
	public List<Patient> getPatient() {
		return patient;
	}
	public void setPatient(List<Patient> patient) {
		this.patient = patient;
	}
	@Override
	public String toString() {
		return "Doctor [did=" + did + ", name=" + name + ", emailid=" + emailid +
", contactno=" + contactno
				+ ", designation=" + designation + ", speciality=" + speciality
+ ", visitinghours=" + visitinghours
				+ ", patient=" + patient + "]";	}}
```

Class Patient

```
package com.example.demo.entity;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.NotBlank;
import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.Range;
import org.springframework.format.annotation.DateTimeFormat;

@Entity
@Table(name = "Appointment_Details")
public class Patient {
	@Id
	@GeneratedValue
	private int aid;
	@Column(name = "PatientName")
	@NotBlank(message="Patient Name cannot be Blank")
	private String pname;
	@Column(name="PatientAge")
	@NotBlank(message="Patient Age cannot be Blank")
	private String age;
	@Column(name="DateOfBirth")
	@Temporal(TemporalType.DATE)
	@DateTimeFormat(pattern = "yyyy-mm-dd")
	private Date dob;
	@Column(name = "Gender")
	private String gender;
	@Column(name = "ContactNumber")
```

```
	@Length(min=10, max=13, message="Mobile number cannot be less than 10 character")
	private String contactno;
	@Column(name="Address")
	@Length(min=3, message="Address value must contains atleast 3 character")
	private String address;
	@Column(name="Reason")
	@NotBlank(message="reason cannot be Blank")
	private String reason;
	@Column(name="Date")
	@Temporal(TemporalType.DATE)
	@DateTimeFormat(pattern = "yyyy-mm-dd")
	private Date appointmentdate;
	@Column(name = "Status")
	private String status;
	@Column(name = "DoctorFee")
	@Range(min =250,message="miminum Fees is 250")
	private double fee;
	public Patient() {
		super();
	}
	public Patient(int aid, @NotBlank(message = "Patient Name cannot be Blank") String pname,
			@NotBlank(message = "Patient Age cannot be Blank") String age, Date dob, String gender,
			@Length(min = 10, max = 13, message = "Mobile number cannot be less than 10 character") String contactno,
			@Length(min = 3, message = "Address value must contains atleast 3 character") String address,
			@NotBlank(message = "reason cannot be Blank") String reason, Date appointmentdate, String status,
			@Range(min = 250, message = "miminum Fees is 250") double fee) {
		super();
		this.aid = aid;
		this.pname = pname;
		this.age = age;
```

```
        this.dob = dob;
        this.gender = gender;
        this.contactno = contactno;
        this.address = address;
        this.reason = reason;
        this.appointmentdate = appointmentdate;
        this.status = status;
        this.fee = fee;
}
public int getAid() {
        return aid;
}
public void setAid(int aid) {
        this.aid = aid;
}
public String getPname() {
        return pname;
}
public void setPname(String pname) {
        this.pname = pname;
}
public String getAge() {
        return age;
}
public void setAge(String age) {
        this.age = age;
}
public Date getDob() {
        return dob;
}
public void setDob(Date dob) {
        this.dob = dob;
}
public String getGender() {
        return gender;
}
public void setGender(String gender) {
```

```
		this.gender = gender;
	}
	public String getContactno() {
		return contactno;
}
	public void setContactno(String contactno) {
		this.contactno = contactno;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
	public String getReason() {
		return reason;
	}
	public void setReason(String reason) {
		this.reason = reason;
	}
	public Date getAppointmentdate() {
		return appointmentdate;
	}
	public void setAppointmentdate(Date appointmentdate) {
		this.appointmentdate = appointmentdate;
	}
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}
	public double getFee() {
		return fee;
	}
	public void setFee(double fee) {
		this.fee = fee;
```

```
	}
	@Override
	public String toString() {
		return "Patient [aid=" + aid + ", pname=" + pname + ", age=" + age + ",
dob=" + dob + ", gender=" + gender
				+ ", contactno=" + contactno + ", address=" + address + ",
reason=" + reason + ", appointmentdate="
				+ appointmentdate + ", status=" + status + ", fee=" + fee +
"]";
	}}
```

Class ErrorMessage

```
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;
		this.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) {
        this.message = message;
}
@Override
public String toString() {
        return "ErrorMessage [status=" + status + ", message=" + message + "]";
}}
```

Application Properties

```
server.port = 8889
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/hospitalms
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
```

Controller Package

Class DoctorController

```
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.Doctor;
import com.example.demo.error.DoctorNotFoundException;
import com.example.demo.service.DoctorService;

@RestController
public class DoctorController {

//Inject One class object to another class using auto wired annotation

	@Autowired
	DoctorService doctorService;

/*
 * Doctor Can
 *		Get All Doctor Record
 *		Add New Doctor
 *		Delete Doctor
 *		Update Doctor
 *		Other Doctor Can See Their Data Using their Id,Name,MailAddress
 */
```

```
//************************ADD DOCTOR************************

	@PostMapping("/AddDoctor/")
	public String doctor(@RequestBody Doctor doctor)
	{
		doctorService.saveDoctor(doctor);
		return "Doctor Record is Inserted Successfully";
	}

//************************GET ALL DOCTOR************************

	@GetMapping("/AllDoctor/")
	public List<Doctor> fetchDoctor()
	{
		return doctorService.fetchDoctorList();
	}

//************************DELETE DOCTOR************************

	@DeleteMapping("/RemoveDoctor/{id}/")
	public String deleteDoctorById(@PathVariable("id")Integer did) throws
DoctorNotFoundException
	{
		doctorService.deleteDoctorById(did);
		return "Doctor is removed from Hospital";
	}

//************************UPDATE DOCTOR************************
```

```
	@PutMapping("/UpdateDoctor/{id}/")
	public String updateDoctor(@PathVariable("id") Integer did, @RequestBody
Doctor doctor) throws DoctorNotFoundException
	{
		doctorService.updateDoctor(did,doctor);
		return "Doctor record is Updated successfully !!";
	}

//********************GET DOCTOR DETAILS BY ID**************

		@GetMapping("/DoctorById/{id}/")
		public Doctor fetchDoctorById(@PathVariable("id") Integer did ) throws
DoctorNotFoundException
		{
			return doctorService.fetchDoctorById(did);

		}

//********************GET DOCTOR DETAILS BY NAME**************

		@GetMapping("/DoctorByName/{name}/")
		public Doctor fetchDoctorByName(@PathVariable("name") String name )
throws DoctorNotFoundException
		{
			return doctorService.fetchDoctorByName(name);
}
```

```
//********************GET DOCTOR DETAILS BY MAIL
ADDRESS**************

		@GetMapping("/DoctorByMail/{mail}/")
		public Doctor fetchDoctorByEmailid(@PathVariable("mail") String
emailid ) throws DoctorNotFoundException
		{
			return doctorService.fetchDoctorByEmailid(emailid);

		}}
```

Class PatientController

```
package com.example.demo.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.entity.Patient;
import com.example.demo.error.PatientNotFoundException;
import com.example.demo.service.PatientService;

@RestController
public class PatientController {

//Inject One class object to Another class using auto wired annotation

	@Autowired
	PatientService patientService;
```

```
/*
 * Patient Can See their Details Using Their Id
 */
//**********************GET PATIENT DETAILS BY ID**************
	@GetMapping("/Patient/{id}/")
	public Patient fetchPatientById(@PathVariable("id")Integer aid ) throws PatientNotFoundException
	{

		return patientService.fetchPatientById(aid);
	}}
```

Class RecepetionistController

```
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.Patient;
import com.example.demo.error.PatientNotFoundException;
import com.example.demo.service.ReceptionistService;
```

```
@RestController
public class ReceptionistController {

	@Autowired
	ReceptionistService receptionistService;

/*
 * Receptionist Can
 *		See All Patient Details
 *		Add Patient Details
 *		Delete All Patient
 *		Update Patient Details
 */

//************************GET ALL PATIENT************************

		@GetMapping("/AllPatient/")
		public List<Patient> fetchPatient()
		{
			return receptionistService.fetchPatientList();
		}
//************************ADD PATIENT*************************

		@PostMapping("/AddPatient/")
		public String savePatient(@RequestBody Patient patient)
		{
			receptionistService.savePatient(patient);
			return "patient record inserted successfully";
		}
```

```
//**********************DELETE PATIENT**********************

        @DeleteMapping("/RemovePatient/{id}/")
        public String deletePatientById(@PathVariable("id") Integer pid) throws
PatientNotFoundException
        {
                receptionistService.deletePatientById(pid);
                return "Patient Cancelled the Appointment";
        }

//**********************UPDATE PATIENT**********************

        @PutMapping("/UpdatePatient/{id}/")
        public Patient updatePatient(@PathVariable("id") Integer pid,
@RequestBody Patient patient) throws PatientNotFoundException
        {
                return receptionistService.updatePatient(pid,patient);
        }

//********************GET PATIENT DETAILS BY NAME*************

        @GetMapping("/PatientByName/{name}/")
        public Patient fetchPatientByPname(@PathVariable("name")String pname
) throws PatientNotFoundException
                {
                        return receptionistService.fetchPatientByPname(pname);

                }}
```

Services Package

Interface DoctorService

```
package com.example.demo.service;
import java.util.List;
import com.example.demo.entity.Doctor;
import com.example.demo.error.DoctorNotFoundException;
public interface DoctorService {

	Doctor saveDoctor(Doctor doctor);

	List<Doctor> fetchDoctorList();

	void deleteDoctorById(Integer did) throws DoctorNotFoundException;

	Doctor updateDoctor(Integer did, Doctor doctor) throws
DoctorNotFoundException;

	Doctor fetchDoctorById(Integer did) throws DoctorNotFoundException;

	Doctor fetchDoctorByName(String name) throws DoctorNotFoundException;

	Doctor fetchDoctorByEmailid(String emailid) throws DoctorNotFoundException;
}
```

Class DoctorServiceImpl

```
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.Doctor;
import com.example.demo.error.DoctorNotFoundException;
import com.example.demo.repository.DoctorRepository;

@Service
public class DoctorServiceImpl implements DoctorService{
@Autowired
	DoctorRepository doctorRepository;

//************************ADD DOCTOR************************
	@Override
	public Doctor saveDoctor(Doctor doctor) {

		return doctorRepository.save(doctor);
	}

//************************GET ALL DOCTOR************************
	@Override
	public List<Doctor> fetchDoctorList() {
		return doctorRepository.findAll();
	}

//************************DELETE DOCTOR************************
	@Override
	public void deleteDoctorById(Integer did) throws DoctorNotFoundException {
```

```
		Optional<Doctor> doc=doctorRepository.findById(did);
		if(!doc.isPresent())
		{
	throw new DoctorNotFoundException("Doctor Id is Not Available Cannot Delete");
		}
		else {
			doctorRepository.deleteById(did);
		}}

//************************UPDATE DOCTOR*************************
	@Override
public Doctor updateDoctor(Integer did, Doctor doctor) throws DoctorNotFoundException {
	Optional<Doctor> doc1=doctorRepository.findById(did);
	Doctor docDB=null;
	if(doc1.isPresent()) {
	 docDB=doctorRepository.findById(did).get();
if(Objects.nonNull(doctor.getName()) && !"".equalsIgnoreCase(doctor.getName()))
{
docDB.setName(doctor.getName());
}
if(Objects.nonNull(doctor.getEmailid()) && !"".equalsIgnoreCase(doctor.getEmailid()))
{
	docDB.setEmailid(doctor.getEmailid())
}
if(Objects.nonNull(doctor.getContactno()) &&
!"".equalsIgnoreCase(doctor.getContactno()))
{
```

```
        docDB.setContactno(doctor.getContactno());
}
if(Objects.nonNull(doctor.getDesignation())                                    &&
!"".equalsIgnoreCase(doctor.getDesignation())) {
      docDB.setDesignation(doctor.getDesignation());
}
if(Objects.nonNull(doctor.getSpeciality())                                     &&
!"".equalsIgnoreCase(doctor.getSpeciality()))
{
   docDB.setSpeciality(doctor.getSpeciality());

}
if(Objects.nonNull(doctor.getVisitinghours())&&!"".equalsIgnoreCase(doctor.getVisiting
hours)) {
   docDB.setVisitinghours(doctor.getVisitinghours());
}
if(Objects.nonNull(doctor.getPatient()) && !"".equals(doctor.getPatient()))
{
   docDB.setPatient(doctor.getPatient());
}
return doctorRepository.save(docDB);
}
else {
    throw new DoctorNotFoundException("Enter Valid Doctor Data to Update the
Record");
}
}
```

```
//************************GET DOCTOR DETAILS BY ID************************
	@Override
public Doctor fetchDoctorById(Integer did) throws DoctorNotFoundException {
				//check for null
Optional<Doctor> doc1= doctorRepository.findById(did);//check in database
if(!doc1.isPresent())
{
   throw new DoctorNotFoundException("Doctor Id is Not available Enter valid ID");
}
else
{
return doctorRepository.findById(did).get();
}
}
//************************GET DOCTOR DETAILS BY NAME**************
	@Override
public Doctor fetchDoctorByName(String name) throws DoctorNotFoundException {
Optional<Doctor> doc2= doctorRepository.findDoctorByName(name);//check in database
if(!doc2.isPresent())
{
   throw new DoctorNotFoundException("Doctor Name is Not available Enter valid Name");
}
else
{
		return doctorRepository.findDoctorByName(name).get();	}
}
```

```
//********************GET DOCTOR DETAILS BY MAIL ADDRESS**************
        @Override
public Doctor fetchDoctorByEmailid(String emailid) throws DoctorNotFoundException
{
Optional<Doctor> doc2= doctorRepository.findDoctorByEmailid(emailid);
if(!doc2.isPresent())
{
    throw new DoctorNotFoundException("Doctor Mail Addresss is not available enter valid Mail Address");
}
else {
        return doctorRepository.findDoctorByEmailid(emailid).get();
    }
}
```

Interface PatientService

```
package com.example.demo.service;
import com.example.demo.entity.Patient;
import com.example.demo.error.PatientNotFoundException;

public interface PatientService {

        Patient fetchPatientById(Integer aid) throws PatientNotFoundException;
}
```

Class PatientServiceImpl

```
package com.example.demo.service;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.demo.entity.Patient;
import com.example.demo.error.PatientNotFoundException;
import com.example.demo.repository.PatientRepository;

@Service
public class PatientServiceImpl implements PatientService{
	@Autowired
	PatientRepository patientRepository;
//******************GET PATIENT DETAILS BY ID****************
	@Override
public Patient fetchPatientById(Integer aid) throws PatientNotFoundException {
			//check for null
Optional<Patient> pat= patientRepository.findById(aid);//check in database
if(!pat.isPresent())
{
	throw new PatientNotFoundException("Patient Id is Not available Enter valid ID");
}
else
{

	return patientRepository.findById(aid).get();
}}}
```

Interface Receptionist Service

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

import java.util.List;
import com.example.demo.entity.Patient;
import com.example.demo.error.PatientNotFoundException;

public interface ReceptionistService {

	void savePatient(Patient patient);

	void deletePatientById(Integer pid) throws PatientNotFoundException;

	Patient updatePatient(Integer pid, Patient patient) throws PatientNotFoundException;
	List<Patient> fetchPatientList();
	Patient fetchPatientByPname(String pname) throws PatientNotFoundException;
}
```

Class ReceptionistServiceImpl

```
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.Patient;
import com.example.demo.error.PatientNotFoundException;
```

```
import com.example.demo.repository.ReceptionistRepository;

@Service
public class ReceptionistServiceImpl implements ReceptionistService{
	@Autowired
	ReceptionistRepository receptionistRepository;
//************************GET ALL PATIENT**********************
		@Override
		 public List<Patient> fetchPatientList() {

			return receptionistRepository.findAll();
		 }
//************************ADD PATIENT**************************
		@Override
		 public void savePatient(Patient patient) {

			receptionistRepository.save(patient);
		 }
//************************DELETE PATIENT**********************
		@Override
public void deletePatientById(Integer aid) throws PatientNotFoundException
{
	Optional<Patient> pat=receptionistRepository.findById(aid);
	if(!pat.isPresent())
	{
	   throw new PatientNotFoundException("Doctor Id is Not Available Cannot Delete");
	}
	else {
```

```
                receptionistRepository.deleteById(aid);
            }
}
//************************UPDATE PATIENT**********************
                @Override
  public Patient updatePatient(Integer aid, Patient patient) throws
PatientNotFoundException {
  Optional<Patient> pat=receptionistRepository.findById(aid);
   Patient patDB=null;
   if(pat.isPresent())
{
 patDB=receptionistRepository.findById(aid).get();
if(Objects.nonNull(patient.getPname()) && !"".equalsIgnoreCase(patient.getPname()))
{
  patDB.setPname(patient.getPname());
}
if(Objects.nonNull(patient.getAge()) && !"".equalsIgnoreCase(patient.getAge()))
{
    patDB.setAge(patient.getAge());
}
if(Objects.nonNull(patient.getDob()) && !"".equals(patient.getDob()))
{
    patDB.setDob(patient.getDob());
}

if(Objects.nonNull(patient.getGender()) && !"".equalsIgnoreCase(patient.getGender()))
{
    patDB.setGender(patient.getGender());
}
```

```
if(Objects.nonNull(patient.getContactno())                                    &&
!"".equalsIgnoreCase(patient.getContactno()))
{
    patDB.setContactno(patient.getContactno());
}
if(Objects.nonNull(patient.getAddress())                                      &&
!"".equalsIgnoreCase(patient.getAddress()))
{
    patDB.setAddress(patient.getAddress());
}
if(Objects.nonNull(patient.getReason()) && !"".equalsIgnoreCase(patient.getReason()))
{
    patDB.setReason(patient.getReason());
}
if(Objects.nonNull(patient.getAppointmentdate())                              &&
!"".equals(patient.getAppointmentdate())) {
    patDB.setAppointmentdate(patient.getAppointmentdate());
}
if(Objects.nonNull(patient.getStatus()) && !"".equalsIgnoreCase(patient.getStatus()))
{
    patDB.setStatus(patient.getStatus());
}
if(Objects.nonNull(patient.getFee()) && !"".equals(patient.getFee()))
{
    patDB.setFee(patient.getFee());
}
return receptionistRepository.save(patient);
}
else
```

```
{
   throw new PatientNotFoundException("Please Enter Valid Data to Update the Record");
}}
//**********************GET PATIENT DETAILS BY NAME*************
        @Override
        public Patient fetchPatientByPname(String pname) throws PatientNotFoundException
{
Optional<Patient> pat=receptionistRepository.findPatientByPname(pname);
if(!pat.isPresent())
{
   throw new PatientNotFoundException("Patient Name is Not Available Cannot Get Details");
}
else
 {
        return receptionistRepository.findPatientByPname(pname).get();
}
}
}
```

Repository Package

Interface ReceptionistRepository

```
package com.example.demo.repository;

import java.util.Optional;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.example.demo.entity.Patient;

@Repository
public interface ReceptionistRepository extends JpaRepository<Patient, Integer>
{
Optional<Patient> findPatientByPname(String pname);
}
```

Interface DoctorRepository

```
package com.example.demo.repository;
import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.entity.Doctor;

@Repository
public interface DoctorRepository extends JpaRepository<Doctor, Integer>
{
        Optional<Doctor> findDoctorByName(String name);
        Optional<Doctor> findDoctorByEmailid(String emailid);
}
```

Interface PatientRepository

```
package com.example.demo.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.example.demo.entity.Patient;

@Repository
public interface PatientRepository extends JpaRepository<Patient, Integer>{

}
```

Error Package

Class DoctorNotFoundException

```
package com.example.demo.error;

public class DoctorNotFoundException extends Exception{

	private static final long serialVersionUID = 1L;

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

Class PatientNotFoundException

```
package com.example.demo.error;
public class PatientNotFoundException extends Exception
{
	private static final long serialVersionUID = -3642012749433417509L;

	public PatientNotFoundException (String s)
	{
		super(s);
	}

}
```

Class 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.ResponseEntityExceptionHand
ler;

import com.example.demo.entity.ErrorMessage;

@ControllerAdvice
```

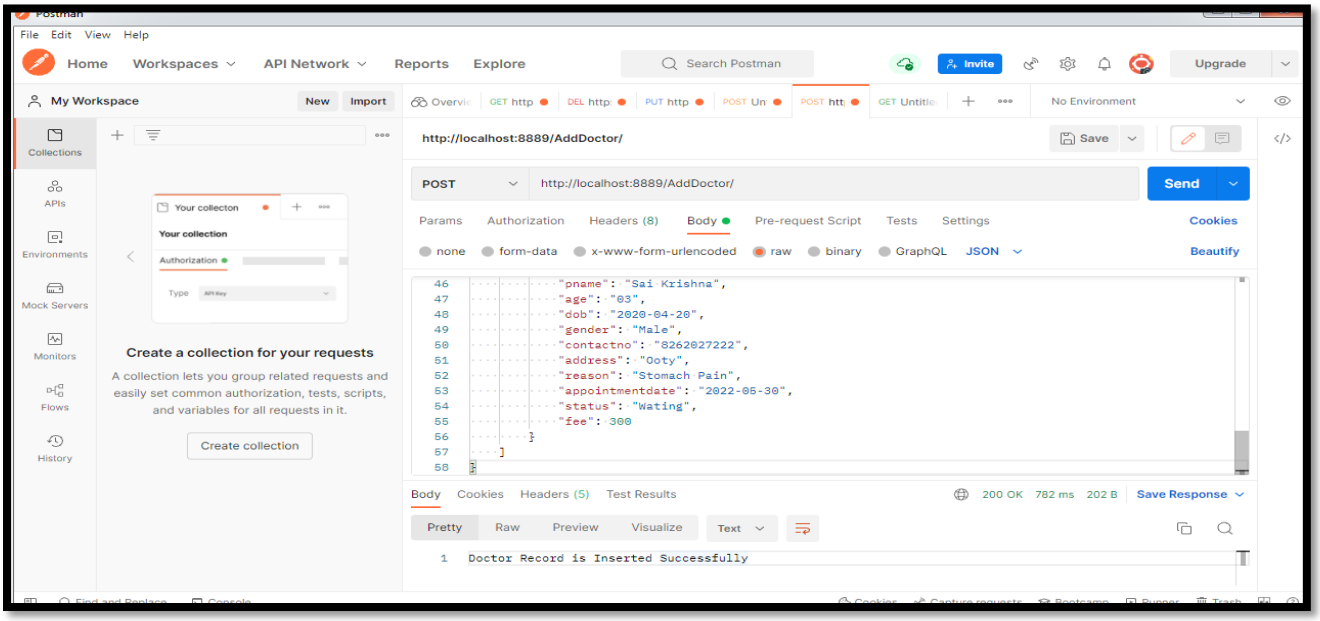
```
@ResponseStatus
public class RestResponseEntityExceptionHandler extends
ResponseEntityExceptionHandler
{
	@ExceptionHandler(DoctorNotFoundException.class)
	public ResponseEntity<ErrorMessage>
departmentNotFoundException(DoctorNotFoundException exception,WebRequest
request)
	{
		ErrorMessage message=new
ErrorMessage(HttpStatus.NOT_FOUND,exception.getMessage());//constructor
	  return ResponseEntity.status(HttpStatus.NOT_FOUND).body(message);
	}

	@ExceptionHandler(PatientNotFoundException.class)
	public ResponseEntity<ErrorMessage>
PatientNotFoundException(PatientNotFoundException exception,WebRequest request) {
		ErrorMessage message=new
ErrorMessage(HttpStatus.NOT_FOUND,exception.getMessage());//constructor
	  return ResponseEntity.status(HttpStatus.NOT_FOUND).body(message);
	}
}
```

Screenshot

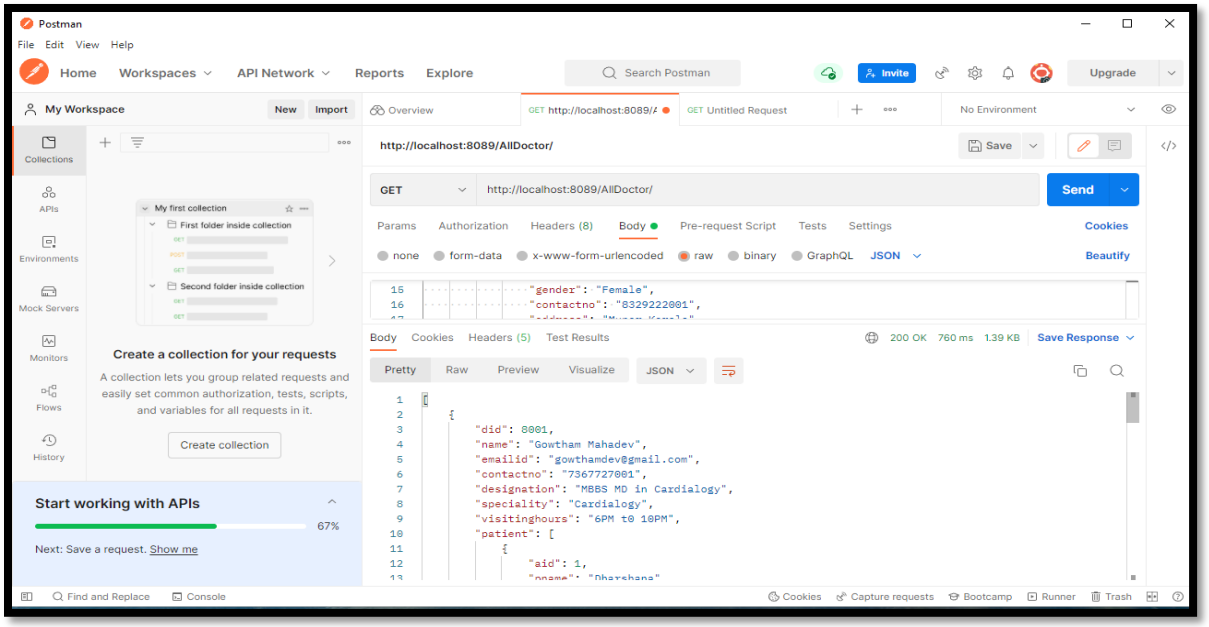
Step 1: Insert Doctor and patient Record

Url: http://localhost:8889/AddDoctor/



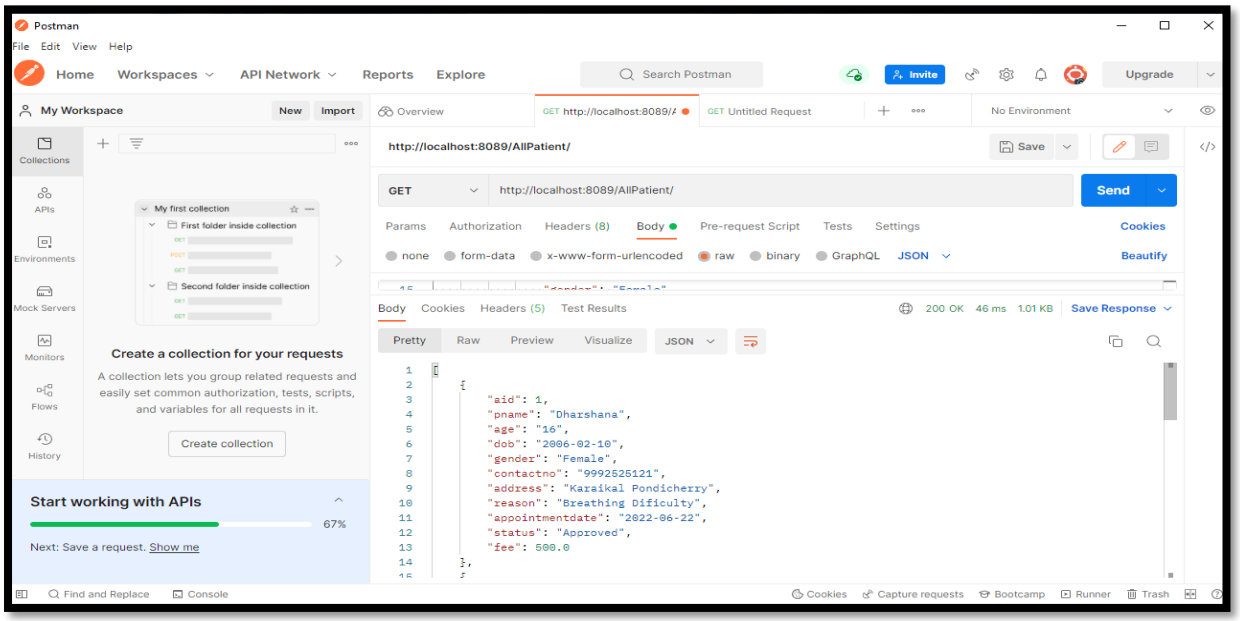
Step 2: Get all Doctor record

Url: http://localhost:8089/AllDoctor/



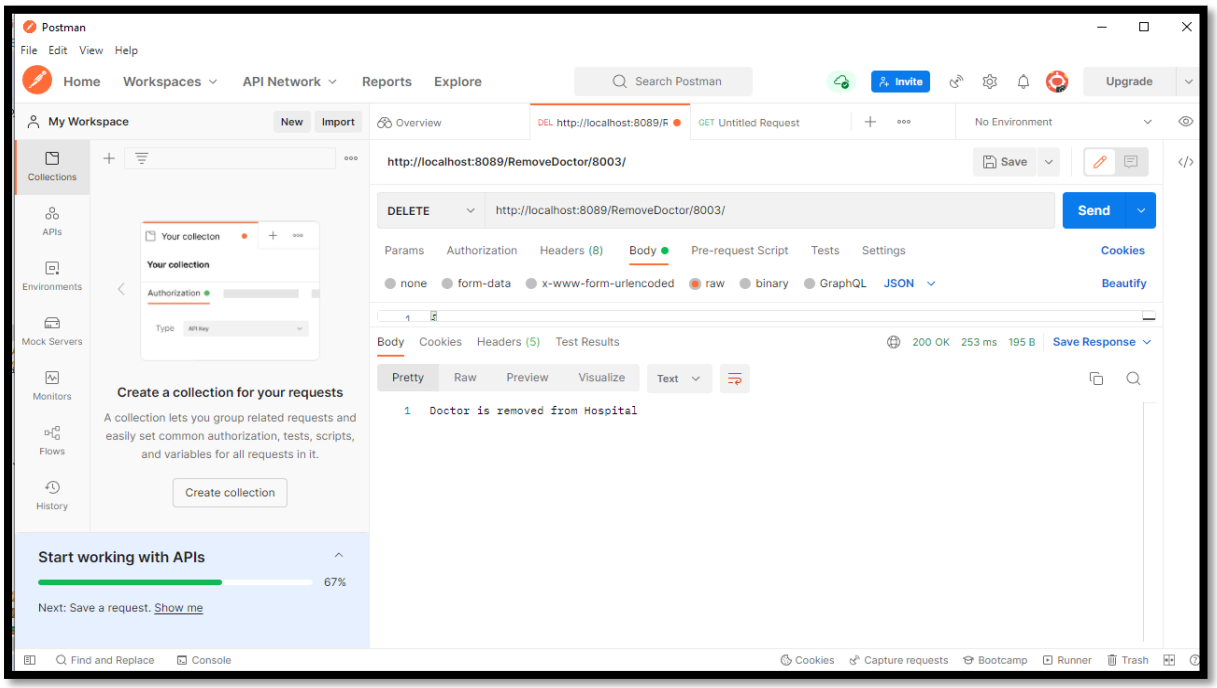
Step 3: Get all patient record

Url: http://localhost:8089/Allpatient/



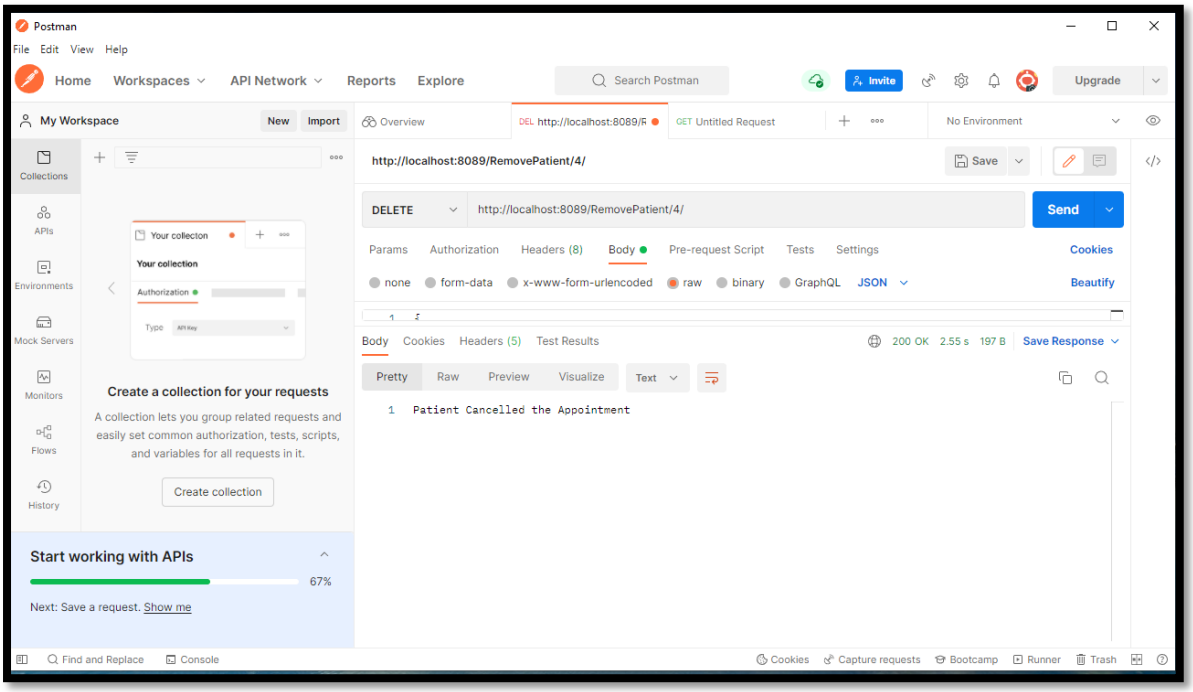
Step 4: Delete the Doctor Record ById

Url: http://localhost:8089/RemoveDoctor/8003/



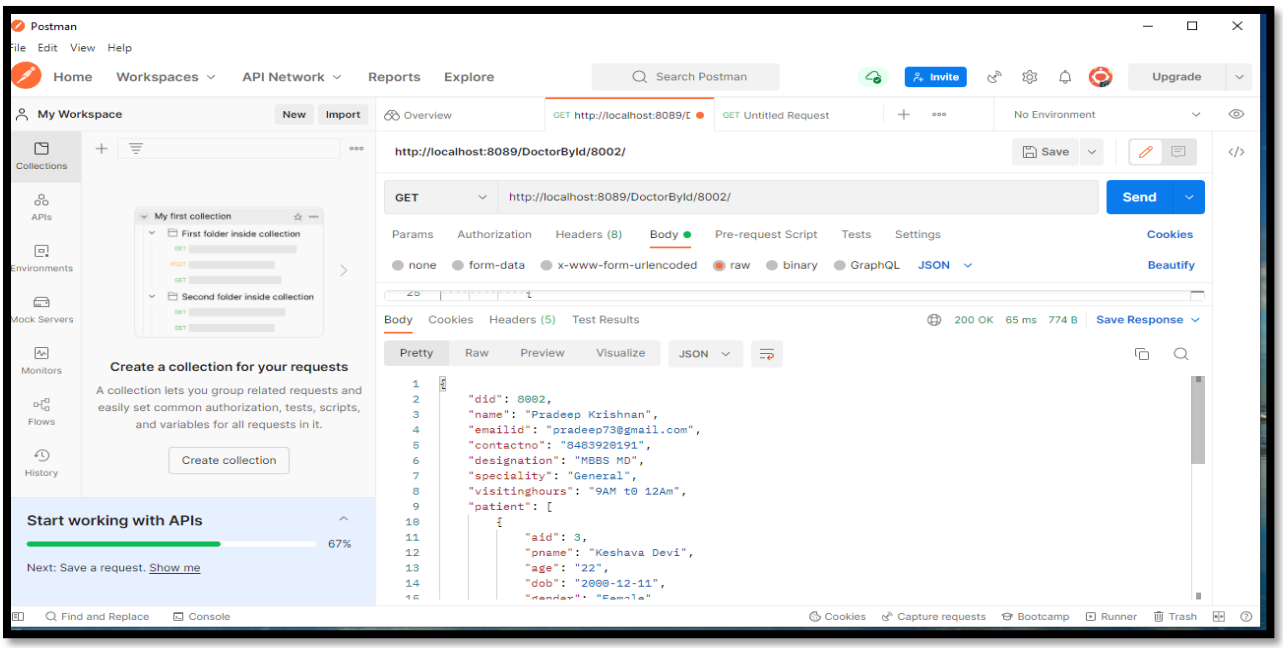
Step 5: Delete Patient Record ById

Url: http://localhost:8089/RemovePatient/4/



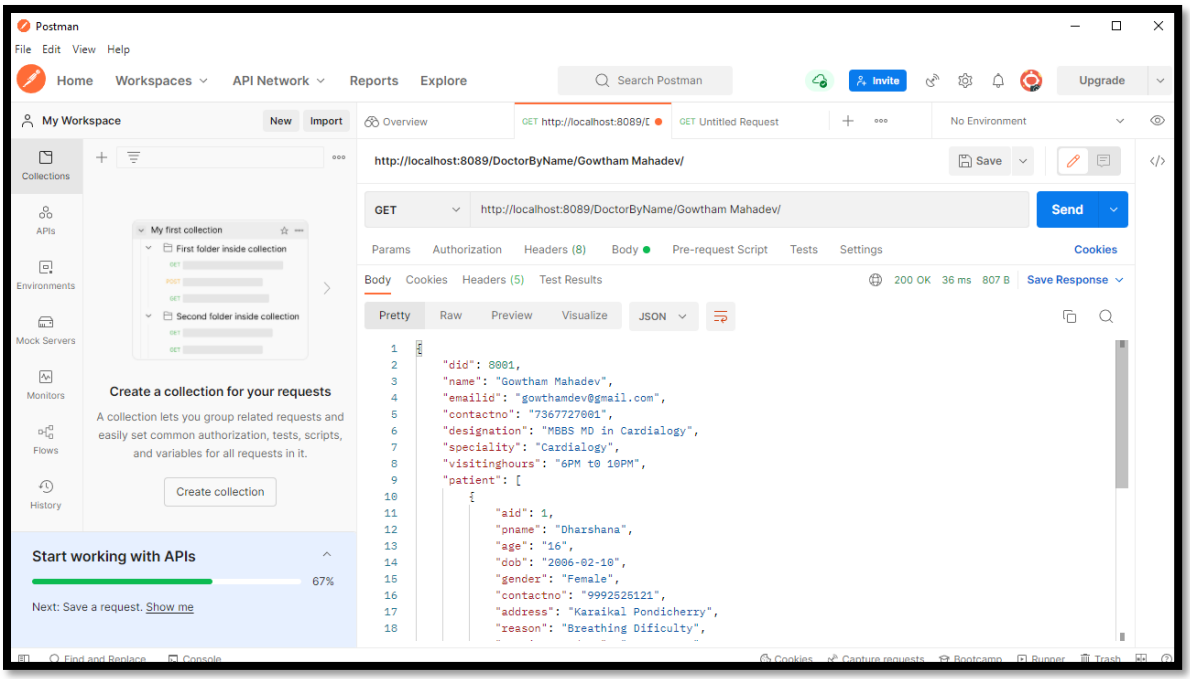
Step 6: Get Doctor Record ById

Url: http://localhost:8089/DoctorById/8002/



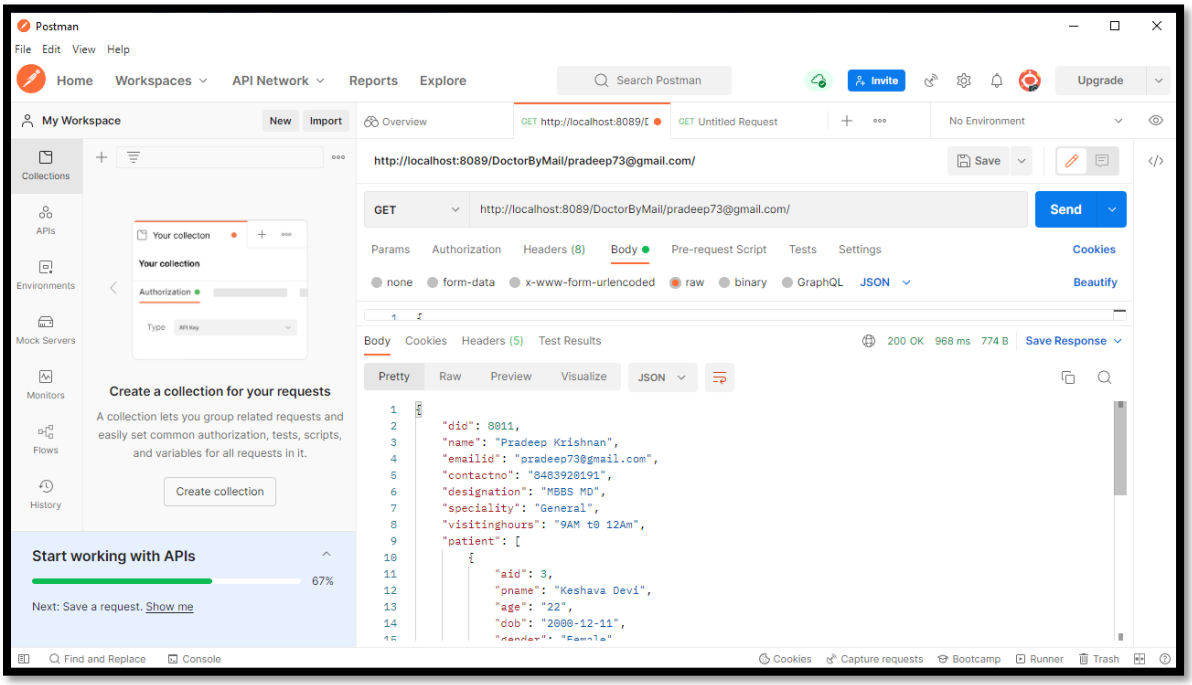
Step 7: Get doctor Record ByName

Url:[http://localhost:8089/lDoctorByName/Gowtham Mahadev/](http://localhost:8089/lDoctorByName/Gowtham%20Mahadev/)



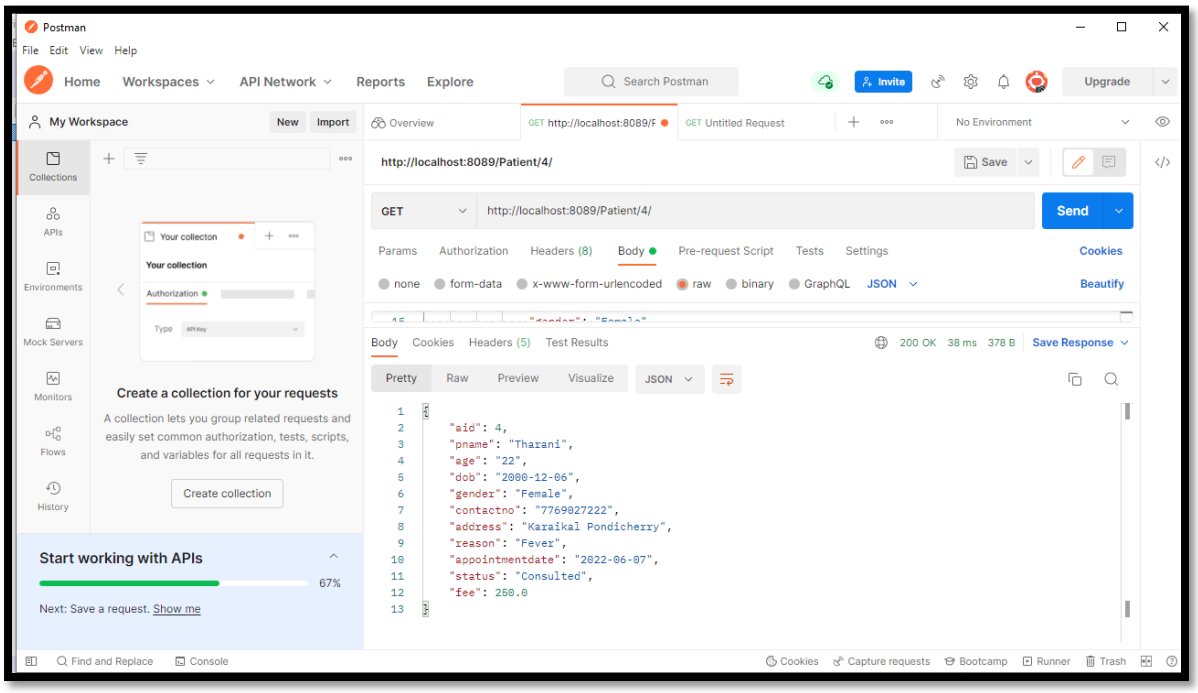
Step 8: Get Doctor Record ByEmailId

Url: http://localhost:8089/DoctorByMail/pradeep73@gmail.com/



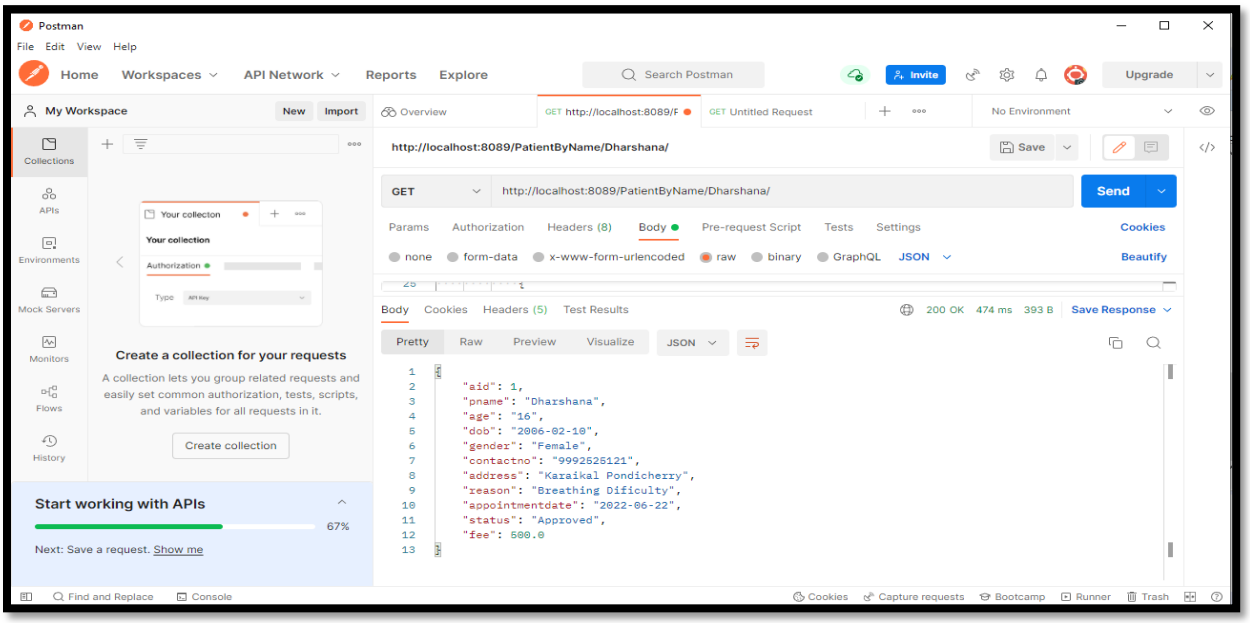
Step 9: Get Patient Record ById

Url: http://localhost:8089/Patient/4/



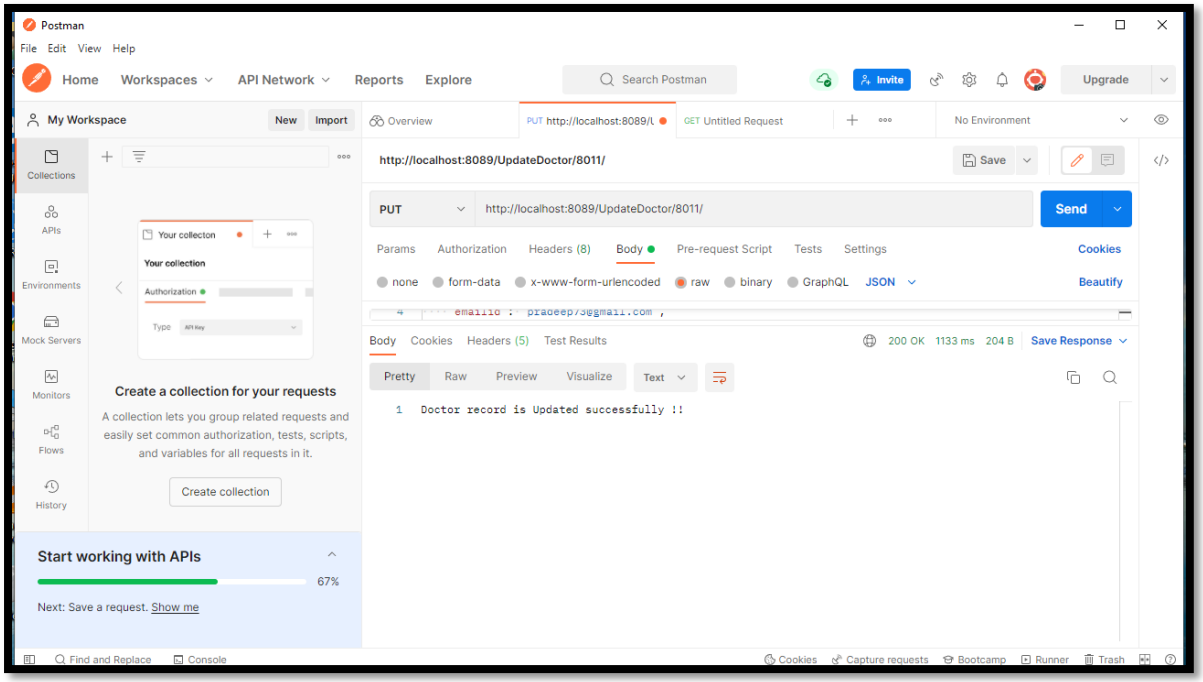
Step 10:Get Patient Record ByName

Url:http://localhost:8089/PatientByName/Dharshana/



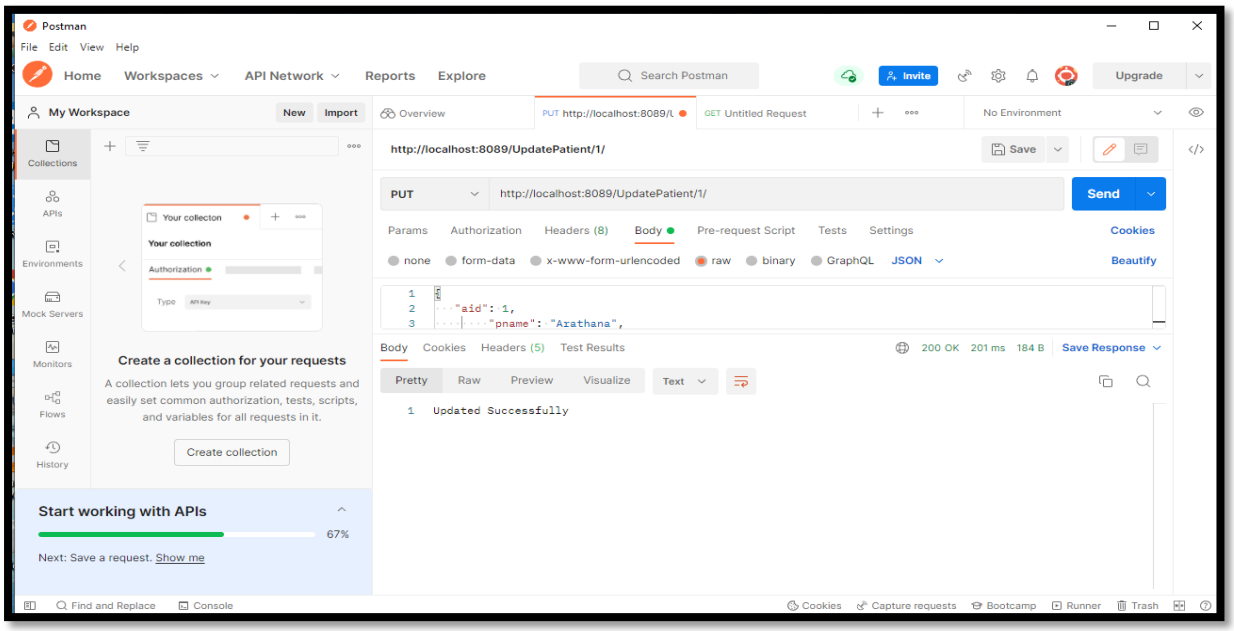
Step 11: Update the Doctor Record ById

Url: http://localhost:8089/UpdateDoctor/8011/



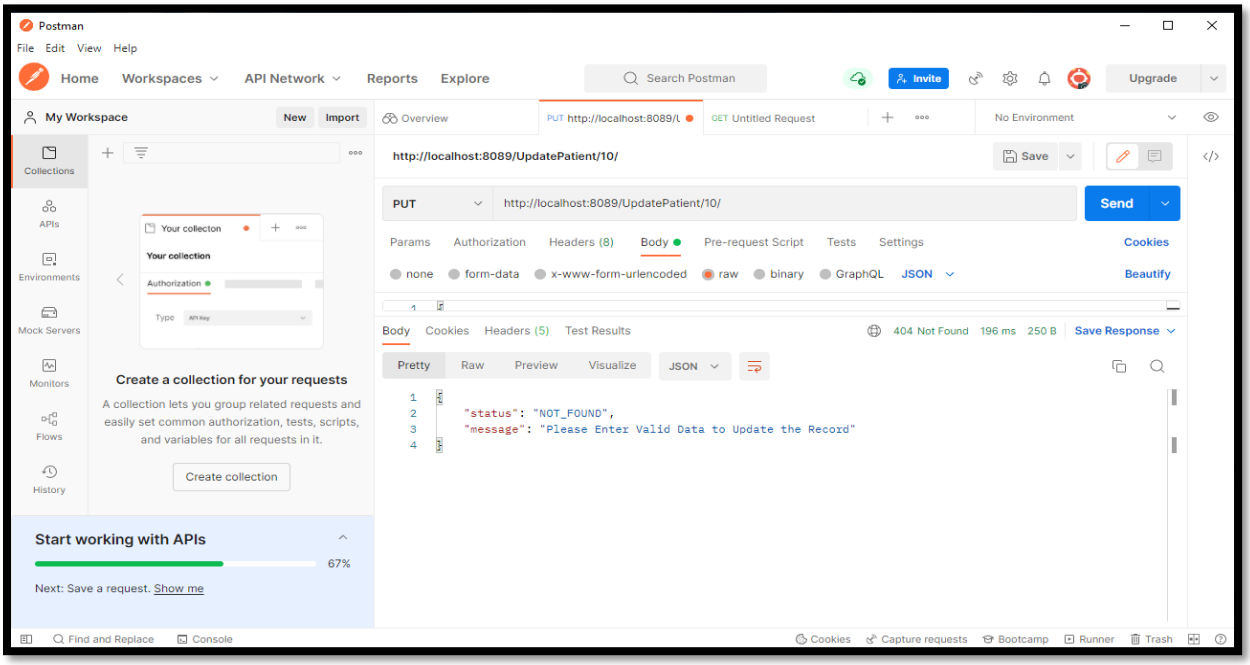
Step 12: Update the Patient Record ById

Url: http://localhost:8089/UpdatePatient/1/



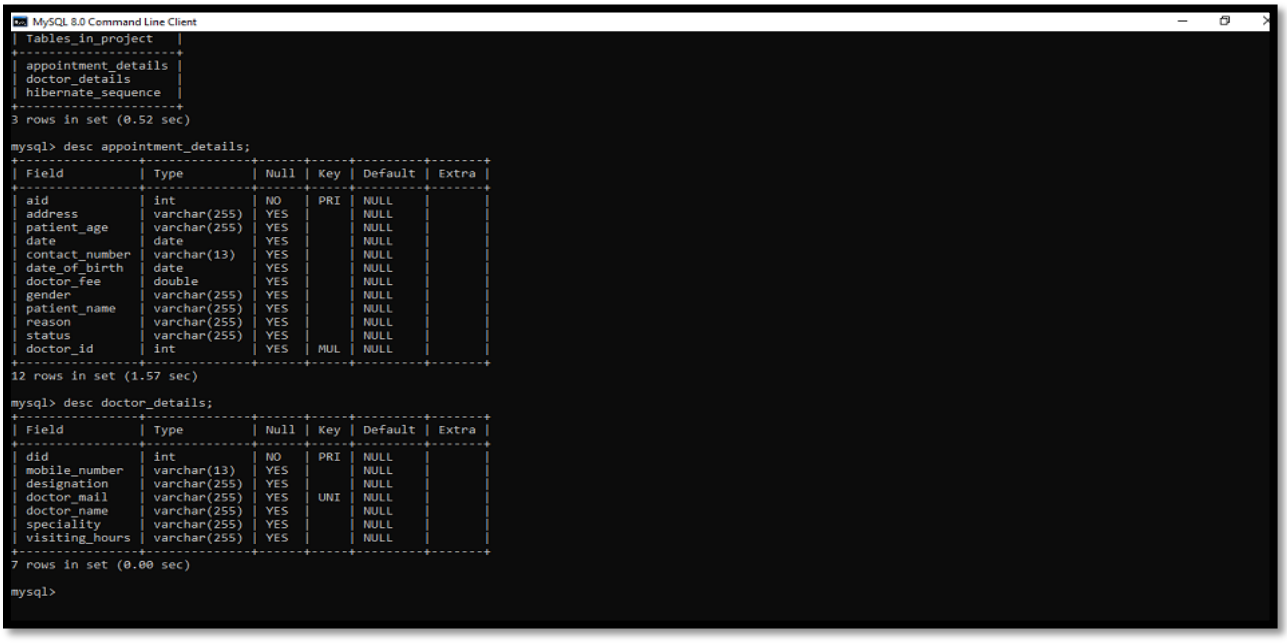
Step 13: If we want to show unavailable patient with their id then it is show exception like “please enter valid data to update the Record”

Url: http://localhost:8089/UpdatePatient/10/



Database Table Design

Doctor and Patient Table



Conclusion

- ✓ Since we are entering details of the patients electronically in the “Hospital Management System”, data will be saved.
- ✓ It easily reduces the book keeping task and thus reduces the human efforts and increases accuracy speed.