A PROJECT REPORT ON STUDENT MANAGEMENT SYSTEM

Submitted By

Ms. ABINAYA P

Ms. GOMATHI G

Ms. MAHALAKSHMI S

Ms. NANDHINI A

Mrs. PAVITHRA S

Batch No. 2021-6769

Under the Guidance of

Trainer Mrs. Indrakka Mali Mam

CONTENTS

1. Introduction

This project "Student Management System" is developed using spring boot framework, which mainly focuses on basic operations of student information system. Like Inserting,Deleting, Updating and getting all records of student's information as well as course details

Course Module:

In the Course Module :

- We can fetch all the course details as well as student details.
- We can get the course details by using course id and Also use course name.
- We can delete the course details by using course id.
- One course studying many students.

Student Module:

In the student module the students can perform:

- Fetch all student's records.
- Fetch student record by student id.
- Fetch student record by student name.
- Delete the record by using student id.

2. Objectives:

- It provides "better and efficient" service".
- Faster way to get information about the students.
- To reduce the unnecessary paper work in maintaining the student information
- All details will be available on a click

3. System Overview:

- The Student management system will be automated the traditional system.
- There is no need to use paper and pen.
- Checking a student details is very easy.
- Adding new student record is very easy.
- Deleting or updating a record of a particular student is simple.

4. Software Requirement:

- Database: MySQL
- API- Spring Data JPA, spring web, Validation, spring security Tools: Postman, IDE-Spring Tool Suite 4
- Coding language-Java 1.8

Spring Tool Suite:

- STS is an IDE (Integrated development environment) to develop spring application
- 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**.**
- It validates our application and provides quick fixes for the application.

Postman:

- Postman is an application used API (Application Programming Interface) testing
- platform to build, test, design, modify, and document APIs.
- It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of response that need to be subsequently validated.
- Postman methods: GET, POST, PUT, DELETE

MySQL:

- MySQL is an open-source relational database management system
- 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

5. Hardware Requirements:

- Edition : Window 10 Pro
- RAM: 8GB
- Processor: AMD PRO A4-4350BR4, 5 COMPUTER CORES 2C+3G 2.50GHZ
- System type: 64-bit operating system

6. Annotations:

***1. @Entity*:**

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

2. @id - @Id annotation specifies the primary key of an entity.

***3. @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 butObject DB also supports this annotation for non-key numeric persistent fields as well.

4. @NOTBLANK- The string is not null and the trimmed length is greater than zero.

5. *@**NOTNULL***- A method should not return null. A variable (like fields, local variables, and parameters) cannot should not hold null value.

***6. @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.

7. @JoinColumn:

JoinColumn annotation makes a column as a join column for an entity association or an element collection.

8. RestController:

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 made by the client.

***9. @Autowired*:**

The spring framework enables automatic dependency injection. In other words, by declaringall the bean dependencies in a spring configuration file, Spring container can autowire

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

10.@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 beandefinitions and service requests for those beans at runtime.

11.@Service:

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

12. @Repository:

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

***13. @PostMapping*:**

The *@PostMapping* is specialized version of @RequestMapping annotation that acts as ashortcut for @RequestMapping(method = RequestMethod.POST).

The @PostMapping annotated methods in the *@Controller* annotated classes handle the HTTP POST requests matched with given URI expression.

***14. @GetMapping*:**

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

15. @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).

16. @ DeleteMapping

@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).

17. @ Exception Handler:

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

18. @SuppressWarnings:

@SuppressWarnings("unchecked") is used when Java generics just don't let you do what you want to, and thus, you need to explicitly specify to the compiler that whatever you are doing is legal and can be executed at the time of execution.

19. @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.

7.Source Code:

```
//STUDENT
package com.example.demo.entity;
import javax.persistence.Column;
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.Range;

@Entity
public class Student {
@Id
@GeneratedValue
private Long sid;
@NotNull(message="Student name cannot be null")
@NotBlank(message="Student name cannot be blank")
private String sname;
@Range(min=20, message="Student age cannot be less than 20 years")
private Integer sage;
@NotBlank(message="Place should not be blank")
private String splace;
@Column(unique=true)
@Email
private String semail;
@Column(length=10)
private String smobile;
```

```
public Student() {
        super();
}

public Student(Long sid,
        @NotNull(message = "Student name cannot be null") @NotBlank(message = "Student name cannot be blank") String sname,
        @Range(min = 20, message = "Student age cannot be less than 20 years") Integer sage,
        @NotBlank(message = "Place should not be blank") String splace, @Email String semail, String smobile) {
        super();
        this.sid = sid;
        this.sname = sname;
        this.sage = sage;
        this.splace = splace;
        this.semail = semail;
        this.smobile = smobile;
}

public Long getSid() {
        return sid;
}

public void setSid(Long sid) {
        this.sid = sid;
}

public String getSname() {
        return sname;
}

public void setSname(String sname) {
```

```
        this.sname = sname;
}

public Integer getSage() {
        return sage;
}

public void setSage(Integer sage) {
        this.sage = sage;
}

public String getSplace() {
        return splace;
}

public void setSplace(String splace) {
        this.splace = splace;
}

public String getSemail() {
        return semail;
}

public void setSemail(String semail) {
        this.semail = semail;
}

public String getSmobile() {
        return smobile;
}

public void setSmobile(String smobile) {
```

```
		this.smobile = smobile;
}

@Override
public String toString() {
		return "Student [sid=" + sid + ", sname=" + sname + ", sage=" + sage + ", splace=" + splace + ", semail=" + semail
			+ ", smobile=" + smobile + "]";
}
}
```

//COURSE

```
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;

@Entity
public class Course {
@Id
@GeneratedValue(generator="seq", strategy=GenerationType.AUTO)
@SequenceGenerator(name="seq", initialValue=1000)
private Long cid;
private String coursename;
private Double coursefees;
```

```
@OneToMany(targetEntity = Student.class, cascade = CascadeType.ALL)
@JoinColumn(name="cid")                    private List<Student> student;



public Course() {
        super();
}

public Course(Long cid, String coursename, Double coursefees, List<Student> student) {
        super();
        this.cid = cid;
        this.coursename = coursename;
        this.coursefees = coursefees;
        this.student = student;
}

public Long getCid() {
        return cid;
}

public void setCid(Long cid) {
        this.cid = cid;
}

public String getCoursename() {
        return coursename;
}

public void setCoursename(String coursename) {
        this.coursename = coursename;
}
```

```
public Double getCoursefees() {
	return coursefees;
}

public void setCoursefees(Double coursefees) {
	this.coursefees = coursefees;
}

public List<Student> getStudent() {
	return student;
}

public void setStudent(List<Student> student) {
	this.student = student;
}

@Override
public String toString() {
	return "Course [cid=" + cid + ", coursename=" + coursename + ", coursefees=" + coursefees + ", student="
+ student
			+ "]";
}
}
```

//STUDENTCONTROLLER

```
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.entity.Student;
import com.example.demo.error.StudentNotFoundException;
import com.example.demo.service.StudentService;

@RestController
public class StudentController {
	@Autowired
	StudentService studentservice;

	//get all record
	@GetMapping("/student/")
	public List<Student> fetchStudentList(){
	return studentservice.fetchStudentList();
	}

	//update record
	@PutMapping("/student/{sid}")
	public Student updateStudent(@PathVariable ("sid") Long sid, @RequestBody Student student) {
	return studentservice.updateStudent(sid, student);
	}

	//delete record
	@DeleteMapping("/student/{sid}")
	public String deleteStudentById(@PathVariable("sid") Long sid) {
	studentservice.deleteStudentById(sid);
	return "Record is deleted";
	}
```

```
        //get the record by name
        @GetMapping("/student/sname/{sname}")
        public Student fetchStudentByName(@PathVariable("sname") String sname) {
        return studentservice.fetchStudentByName(sname);
        }

        //get record by id, if id is not present then print error message
        @GetMapping("/student/{sid}")
        public Student fetchStudentById(@PathVariable("sid") Long sid) throws StudentNotFoundException {
        return studentservice.fetchStudentById(sid);
        }
}
```

//STUDENTSERVICE

```
package com.example.demo.service;
import java.util.List;
import com.example.demo.entity.Student;
import com.example.demo.error.StudentNotFoundException;
public interface StudentService {

        //get all record
        List<Student> fetchStudentList();

        //update
        Student updateStudent(Long sid, Student student);

        //delete
        void deleteStudentById(Long sid);

        //get record by name
```

```
        Student fetchStudentByName(String sname);


        //get record by id, if id is not present then print error message
        Student fetchStudentById(Long sid) throws StudentNotFoundException;
}
```

```
//STUDENTSERVICEIMPL


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.Student;
import com.example.demo.error.StudentNotFoundException;
import com.example.demo.repository.StudentRepository;
@Service
public class StudentServiceImpl implements StudentService {
        @Autowired
        StudentRepository studentRepo;


        //get record
        @Override
        public List<Student> fetchStudentList() {
        return studentRepo.findAll();
        }


        //update record
        @SuppressWarnings("unlikely-arg-type")
        @Override
        public Student updateStudent(Long sid, Student student) {
```

```
Optional<Student> student1= studentRepo.findById(sid);
Student stuDB=null;
if(student1.isPresent()) {
	stuDB=studentRepo.findById(sid).get();
	if(Objects.nonNull(student.getSname()) && !"".equalsIgnoreCase(student.getSname())) {
		stuDB.setSname(student.getSname());
	}
	if(Objects.nonNull(student.getSage()) && !"".equals(student.getSage())) {
		stuDB.setSage(student.getSage());
		System.out.println(student.getSage());
	}
	if(Objects.nonNull(student.getSplace()) && !"".equalsIgnoreCase(student.getSplace())) {
		stuDB.setSplace(student.getSplace());
		System.out.println(student.getSplace());
	}
	if(Objects.nonNull(student.getSemail()) && !"".equalsIgnoreCase(student.getSemail())) {
		stuDB.setSemail(student.getSemail());
		System.out.println(student.getSemail());
	}
	if(Objects.nonNull(student.getSmobile()) && !"".equalsIgnoreCase(student.getSmobile())) {
		stuDB.setSmobile(student.getSmobile());
		System.out.println(student.getSmobile());
	}
}
return studentRepo.save(stuDB);
}

//delete
@Override
public void deleteStudentById(Long sid) {
studentRepo.deleteById(sid);
}
```

```
	//get record by name
	@Override
	public Student fetchStudentByName(String sname) {
	return studentRepo.findBySname(sname);
	}

	//get record by id, if id is not present then print error message
	@Override
	public Student fetchStudentById(Long sid) throws StudentNotFoundException {
	Optional<Student> student1= studentRepo.findById(sid);//check in database
	if(!student1.isPresent()) {
		throw new StudentNotFoundException("Student not available");
	}
	return studentRepo.findById(sid).get();
	}
}
```

//STUDENTREPOSITORY

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

@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {

	//get by name
	public Student findBySname(String sname);
}
```

```
//COURSECONTROLLER

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.Course;
import com.example.demo.error.CourseNotFoundException;
import com.example.demo.service.CourseService;

@RestController
public class CourseController {
	@Autowired
	CourseService courseservice;

	//insert
	@PostMapping("/course/")
	public Course SaveCourse(@RequestBody Course course) {
	return courseservice.SaveCourse(course);
	}

	//get all record
	@GetMapping("/course/")
	public List<Course> fetchCourseList(){
	return courseservice.fetchCourseList();
```

```
}

//update record
@PutMapping("/course/{cid}")
public Course updateCourse(@PathVariable ("cid") Long cid, @RequestBody Course course) {
return courseservice.updateCourse(cid, course);
}

//delete record
@DeleteMapping("/course/{cid}")
public String deleteCourseById(@PathVariable("cid") Long cid) {
courseservice.deleteCourseById(cid);
return "Record is deleted";
}

//get the record by course name
@GetMapping("/course/coursename/{coursename}")
public Course fetchCourseByName(@PathVariable("coursename") String coursename) {
return courseservice.fetchCourseByName(coursename);
}

//get the record by course fees
@GetMapping("/course/coursefees/{coursefees}")
public Course fetchCourseByFees(@PathVariable("coursefees") Double coursefees) {
return courseservice.fetchCourseByFees(coursefees);
}

//get record by id, if id is not present then print error message
@GetMapping("/course/{cid}")
public Course fetchCourseById(@PathVariable("cid") Long cid) throws CourseNotFoundException {
return courseservice.fetchCourseById(cid);
}
```

```
}
```

```
//COURSESERVICE

package com.example.demo.service;
import java.util.List;
import com.example.demo.entity.Course;
import com.example.demo.error.CourseNotFoundException;

public interface CourseService {
        //post
        Course SaveCourse(Course course);

        //get record
        List<Course> fetchCourseList();

        //delete
        void deleteCourseById(Long cid);

        //update
        Course updateCourse(Long cid, Course course);

        //get record by name
        Course fetchCourseByName(String coursename);

        //get record by fees
        Course fetchCourseByFees(Double coursefees);

        //get record by id, if id is not present then print error message
        Course fetchCourseById(Long cid) throws CourseNotFoundException;
```

```
}
```

```
//COURSESERVICEIMPL

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.Course;
import com.example.demo.error.CourseNotFoundException;
import com.example.demo.repository.CourseRepository;

@Service
public class CourseServiceImpl implements CourseService {
	@Autowired
	CourseRepository courseRepo;

	//post
	@Override
	public Course SaveCourse(Course course) {
	return courseRepo.save(course);
	}

	//get record
	@Override
	public List<Course> fetchCourseList() {
	return courseRepo.findAll();
	}
```

```
//update record
@SuppressWarnings("unlikely-arg-type")
@Override
public Course updateCourse(Long cid, Course course) {
Optional<Course> course1= courseRepo.findById(cid);
Course crsDB=null;
if(course1.isPresent()) {
	crsDB=courseRepo.findById(cid).get();
	if(Objects.nonNull(course.getCoursename()) && !"".equalsIgnoreCase(course.getCoursename())) {
		crsDB.setCoursename(course.getCoursename());
	}
	if(Objects.nonNull(course.getCoursefees()) && !"".equals(course.getCoursefees())) {
		crsDB.setCoursefees(course.getCoursefees());
		System.out.println(course.getCoursefees());
	}

}
return courseRepo.save(crsDB);
}

//delete
@Override
public void deleteCourseById(Long cid) {
courseRepo.deleteById(cid);
}

//get record by name
@Override
public Course fetchCourseByName(String coursename) {
return courseRepo.findByCoursename(coursename);
}
```

```
	//get record by fees
	@Override
	public Course fetchCourseByFees(Double coursefees) {
	return courseRepo.findByCoursefees(coursefees);
	}

	//get record by id, if id is no present then print error message
	@Override
	public Course fetchCourseById(Long cid) throws CourseNotFoundException {
	//check for null
	Optional<Course> course1= courseRepo.findById(cid);//check in database
	if(!course1.isPresent()) {
		throw new CourseNotFoundException("Course not available");
	}
	return courseRepo.findById(cid).get();
	}
}
```

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

@Repository
public interface CourseRepository extends JpaRepository<Course, Long> {

	//get record by course name
	Course findByCoursename(String coursename);

	//get record by course fees
```

```
	Course findByCoursefees(Double coursefees);
}
```

//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 + "]";
	}
	}
```

//STUDENTNOTFOUNDEXCEPTION

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

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

//COURSENOTFOUNDEXCEPTION

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

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

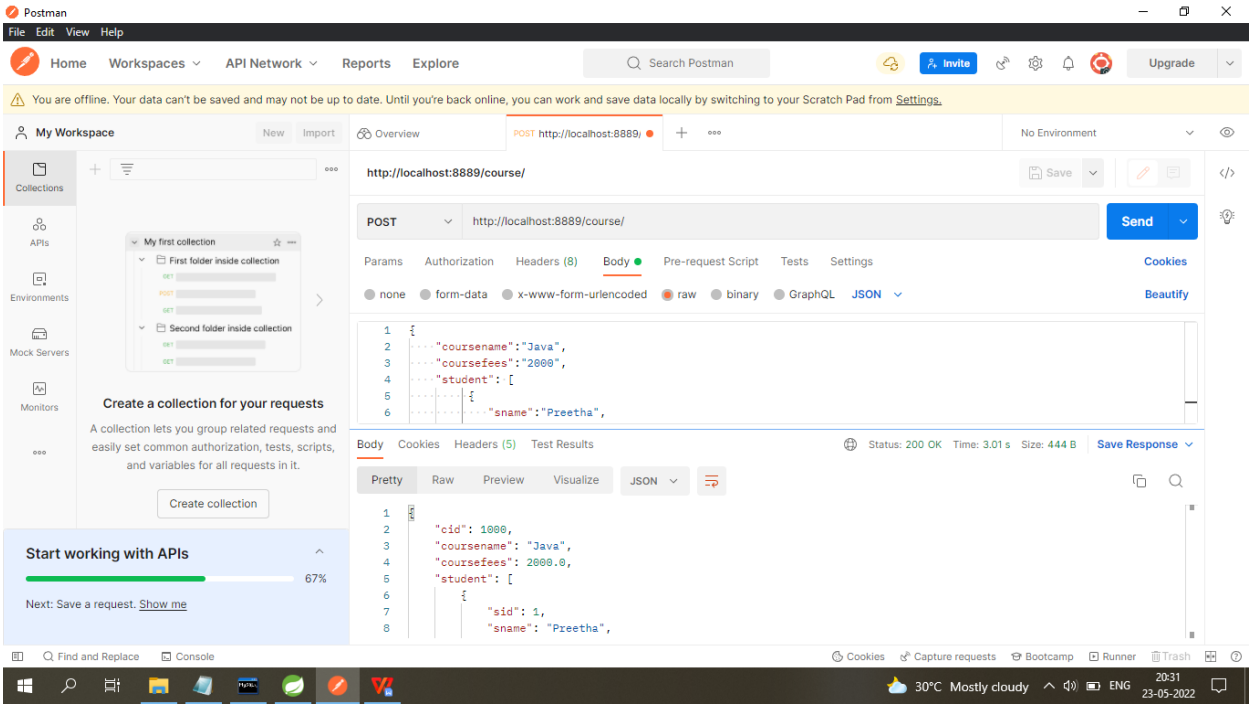
```
}
```

```
//REQUESTRESPONSEENTITYECEPTIONHANDLER


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 RequestResponseEntityExceptionHandler extends ResponseEntityExceptionHandler{

           @ExceptionHandler(StudentNotFoundException.class)

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

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

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

           }


         @ExceptionHandler(CourseNotFoundException.class)

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

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

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

           }

        }
```

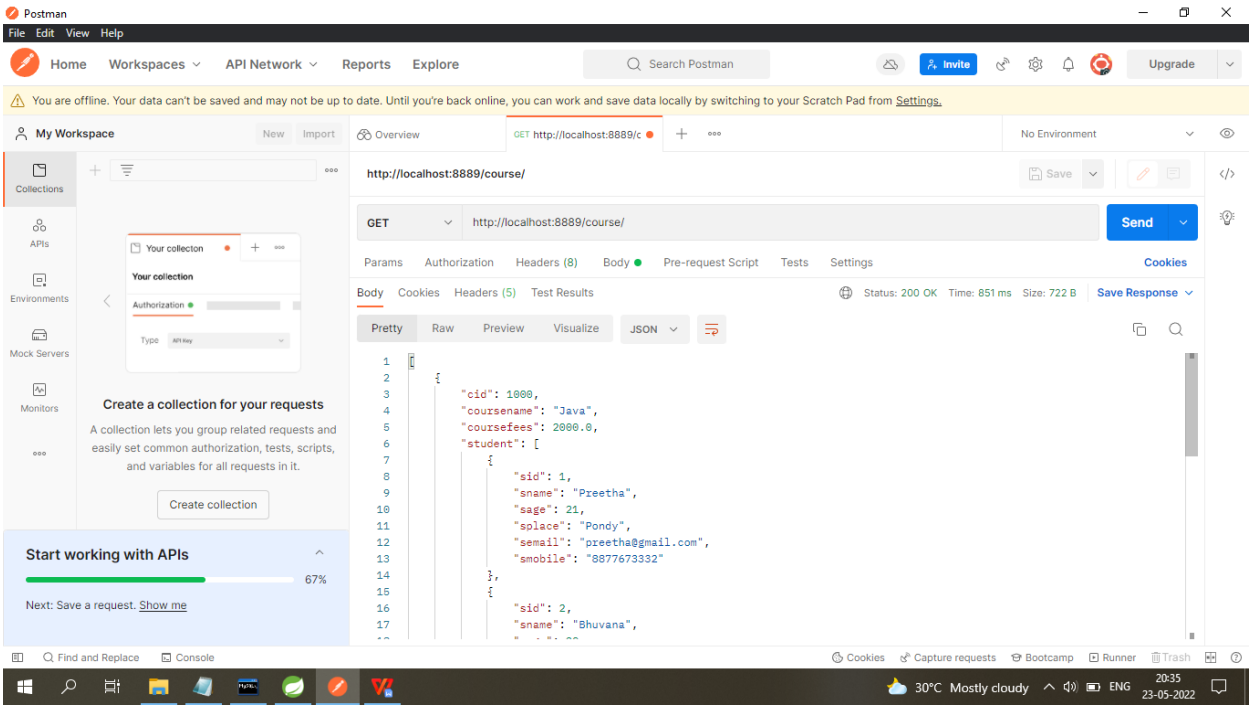
8. SCREENSHOTS:

COURSE

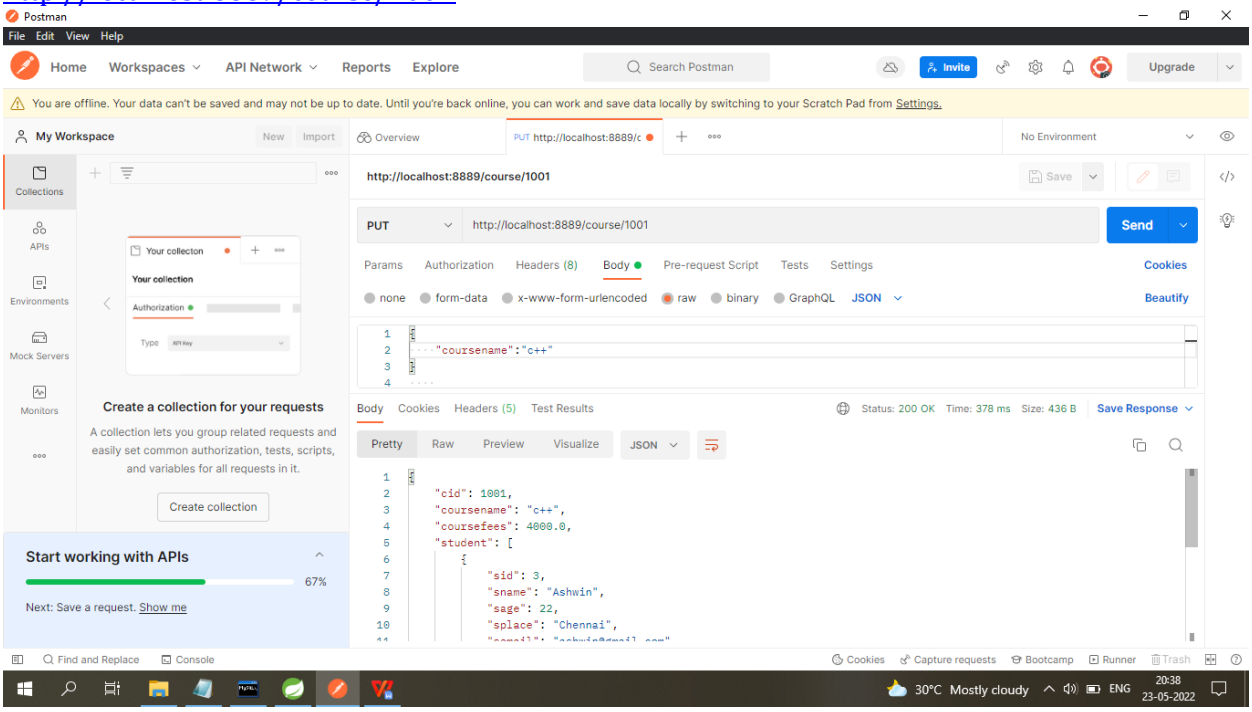
Step 1: Insert record
http://localhost:8889/course/



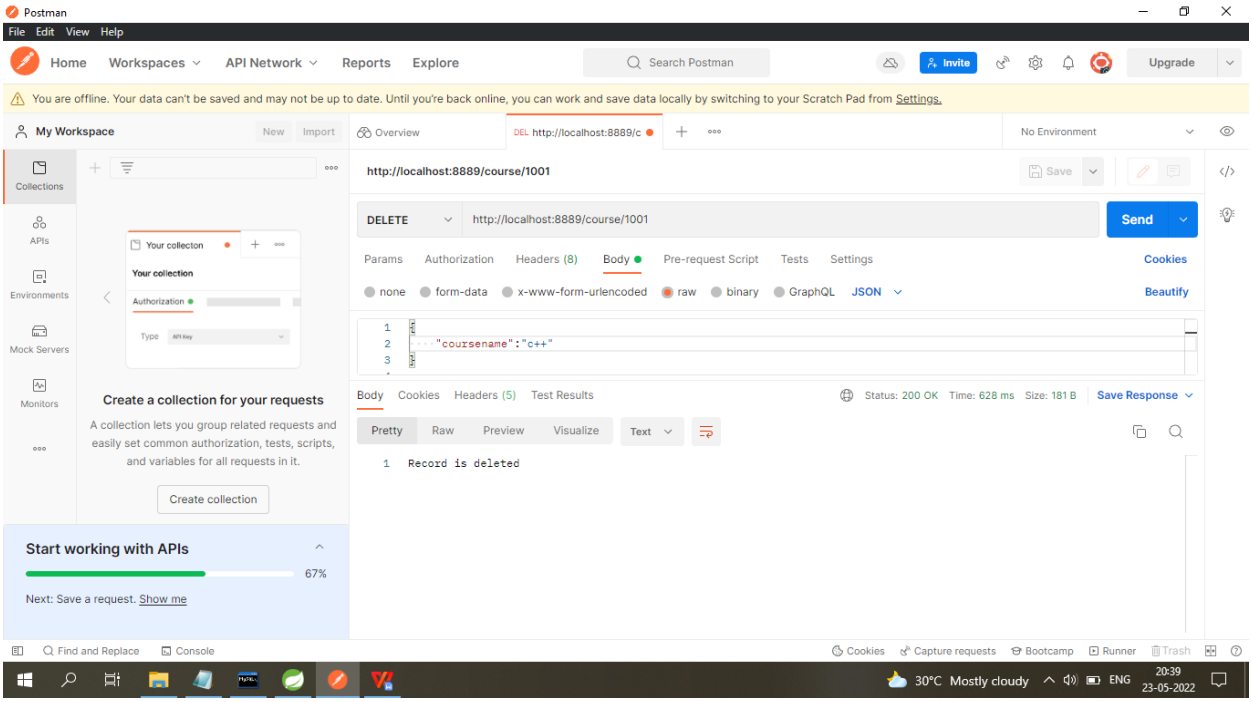
Step 2: Get all course record
http://localhost:8889/course/



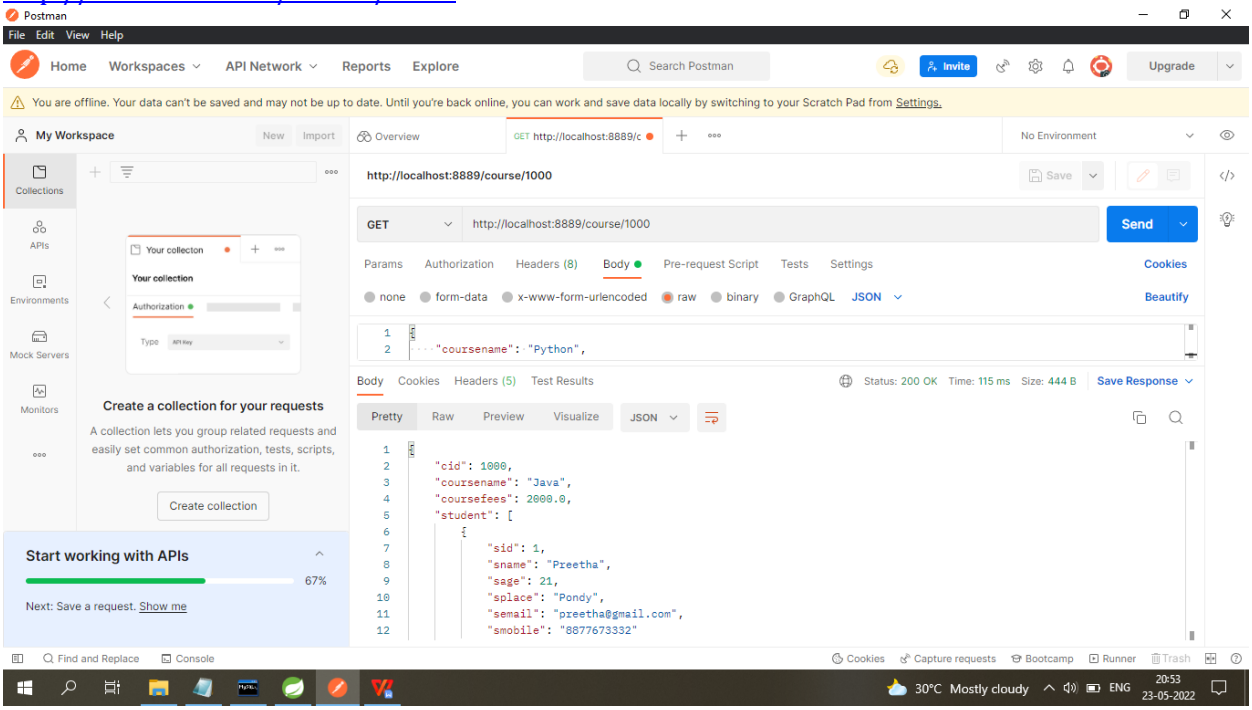
Step 3: Update course record using course id
http://localhost:8889/course/1001



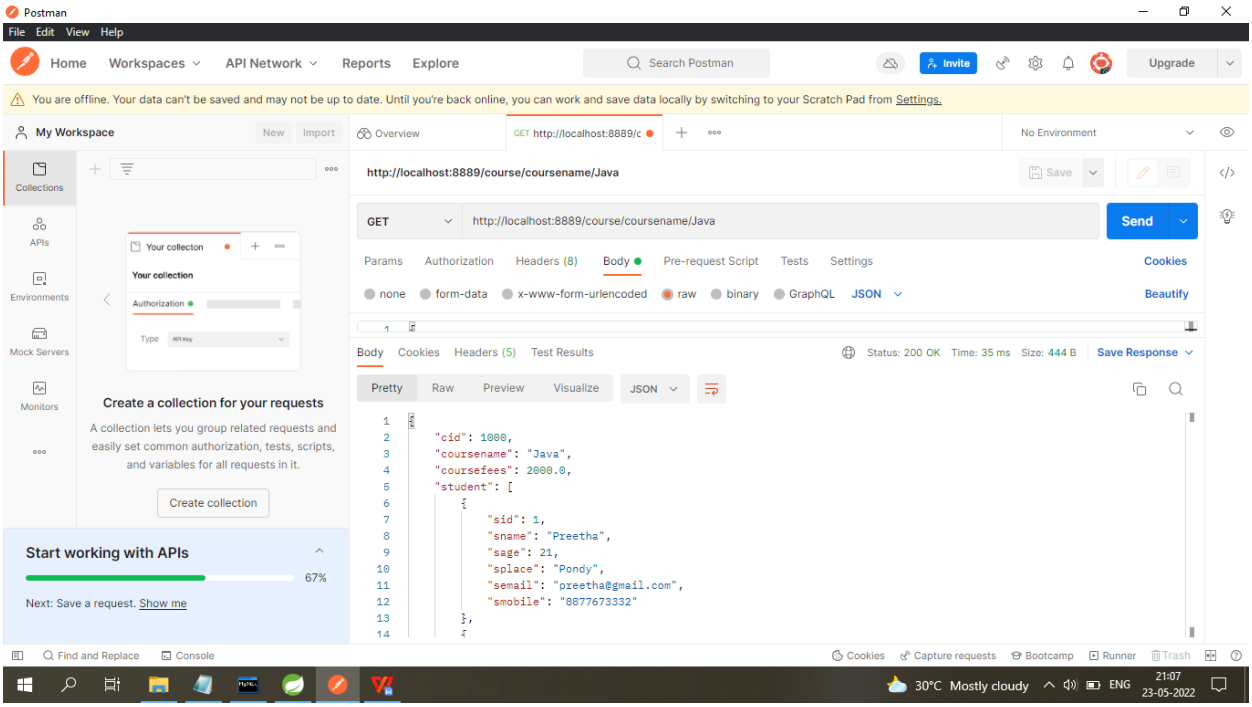
Step 4: Delete course record using course id
http://localhost:8889/course/1001



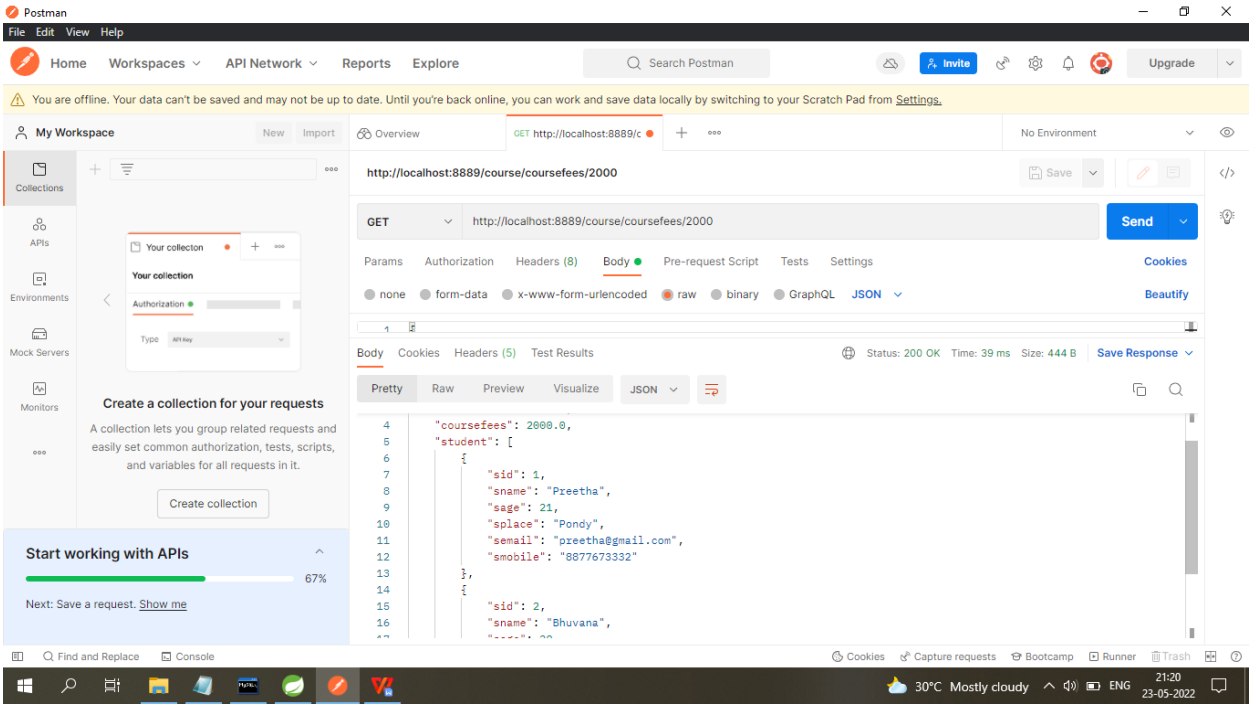
Step 5: Get record by using course id
http://localhost:8889/course/1000



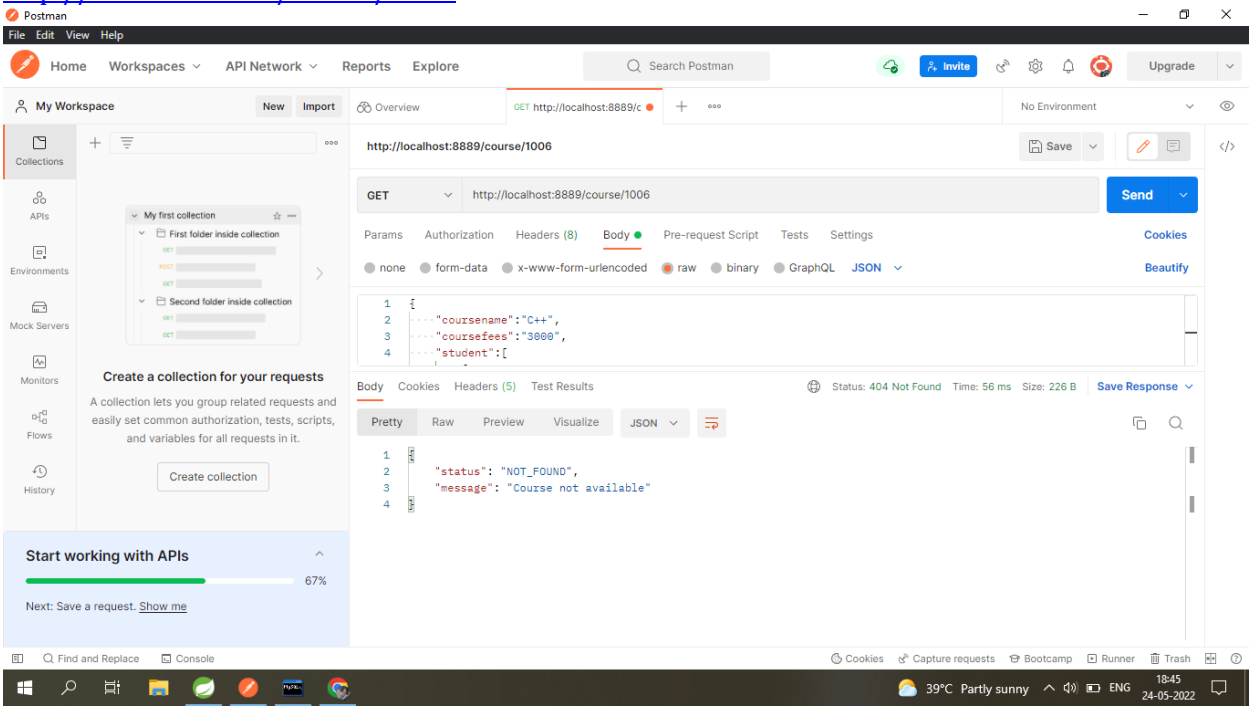
Step 6: Get record by using course name
http://localhost:8889/coursename/java



Step7: Get record by using Course Fees
http://localhost:8889/course/coursefees/2000

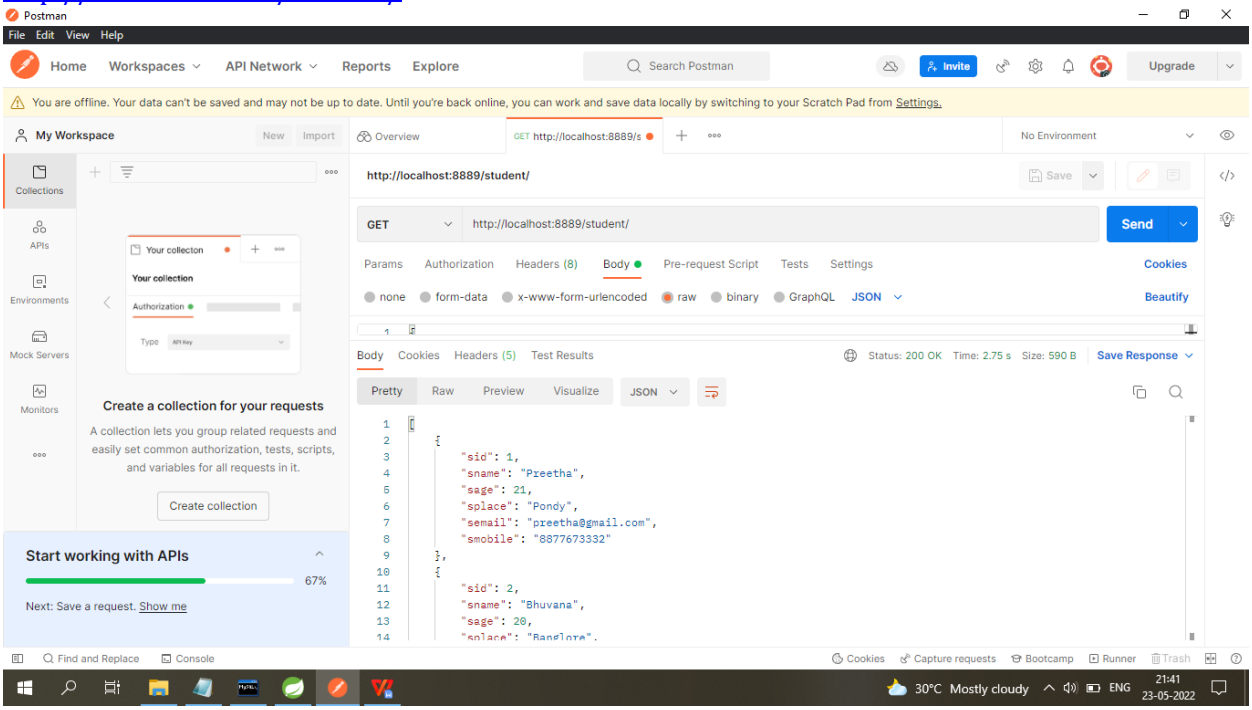


Step 8: Get unknown course record using course id, it will print error message
http://localhost:8889/course/1006

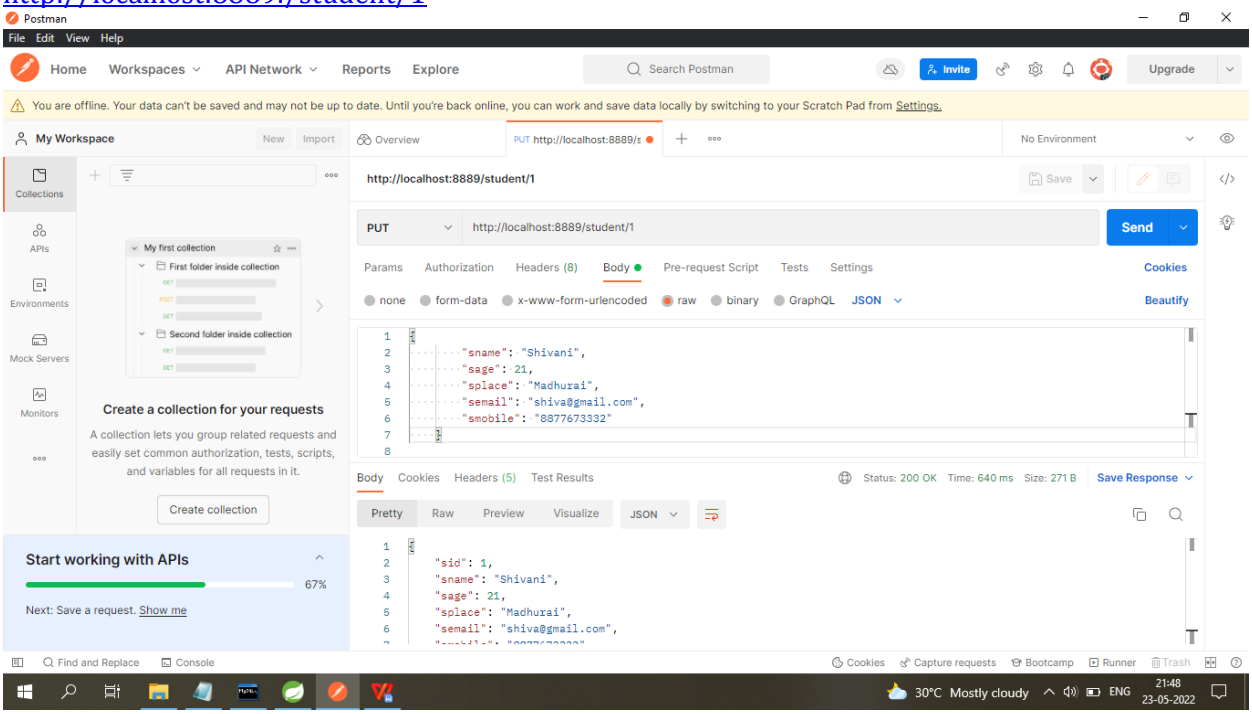


STUDENT

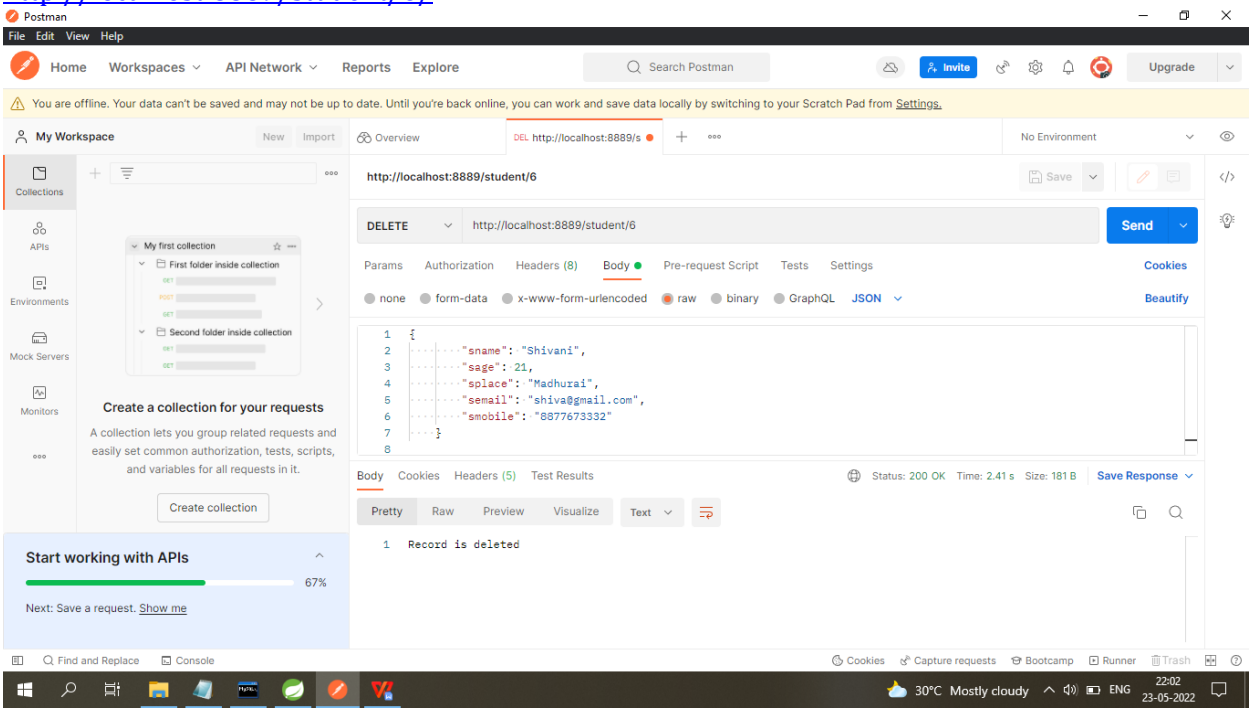
Step 9: Get record from student
http://localhost:8889/student/



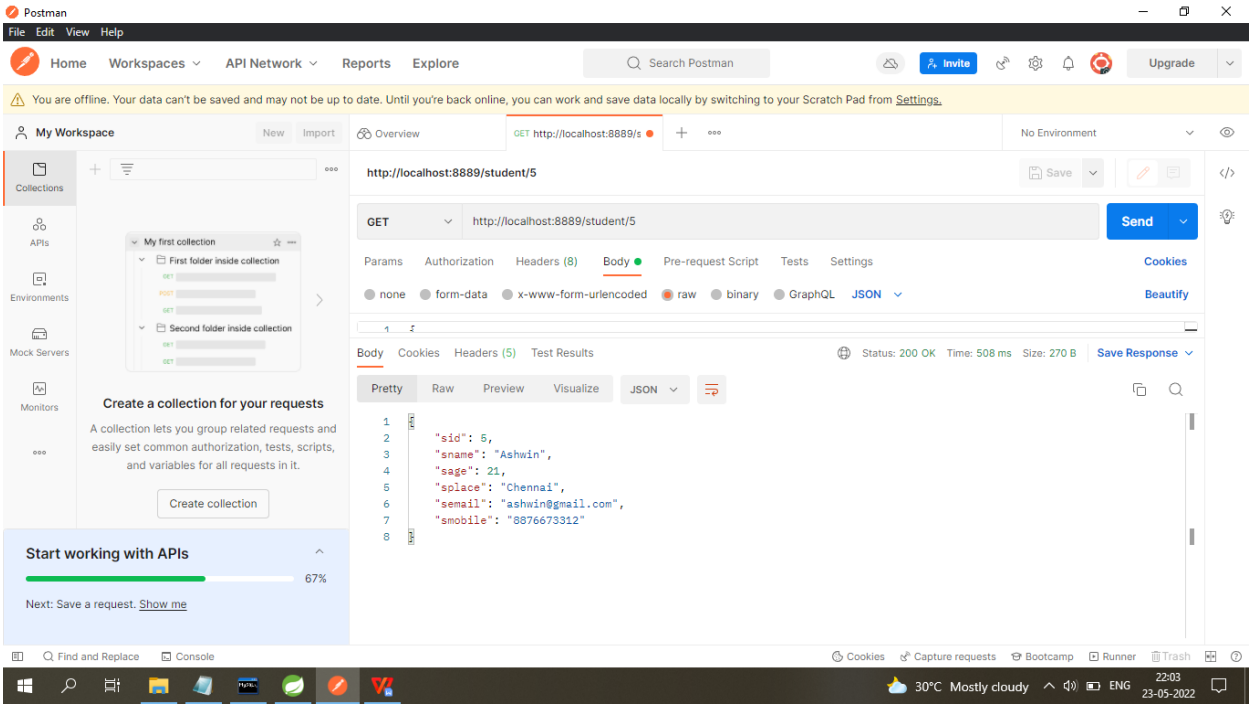
Step 10: update record using student id
http://localhost:8889:/student/1



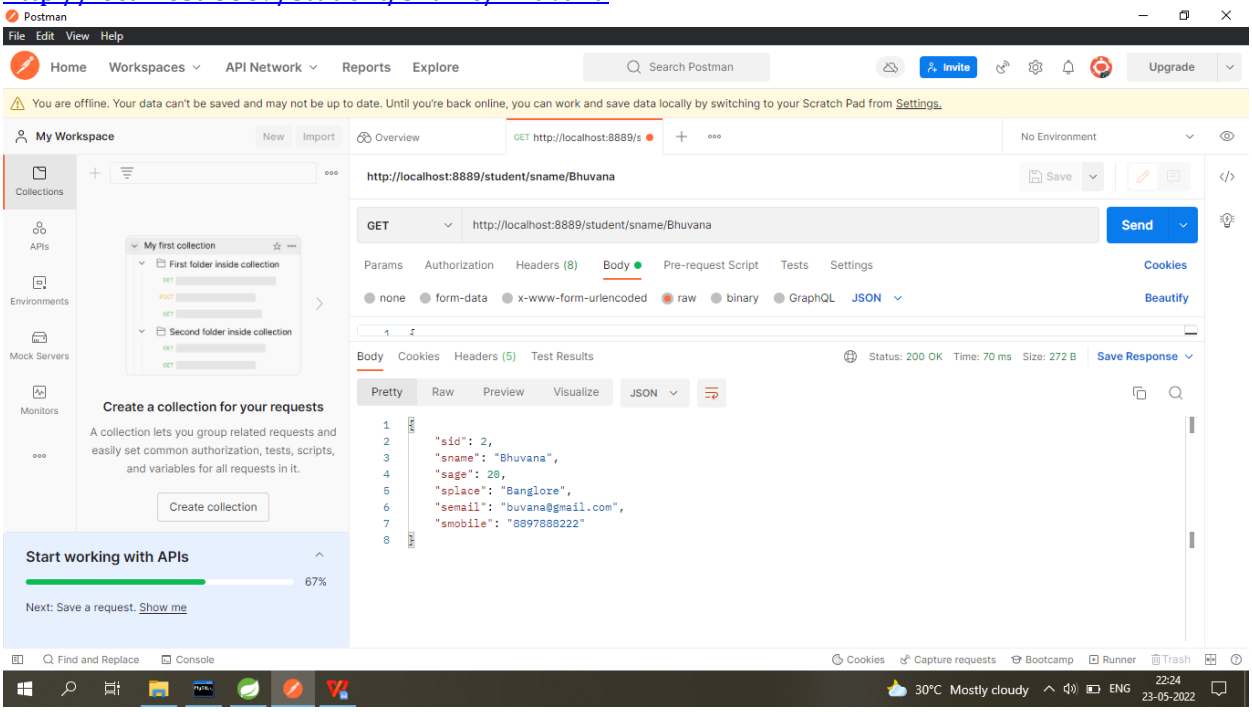
Step 11: delete student record by using student id
http://localhost:8889/student/6/



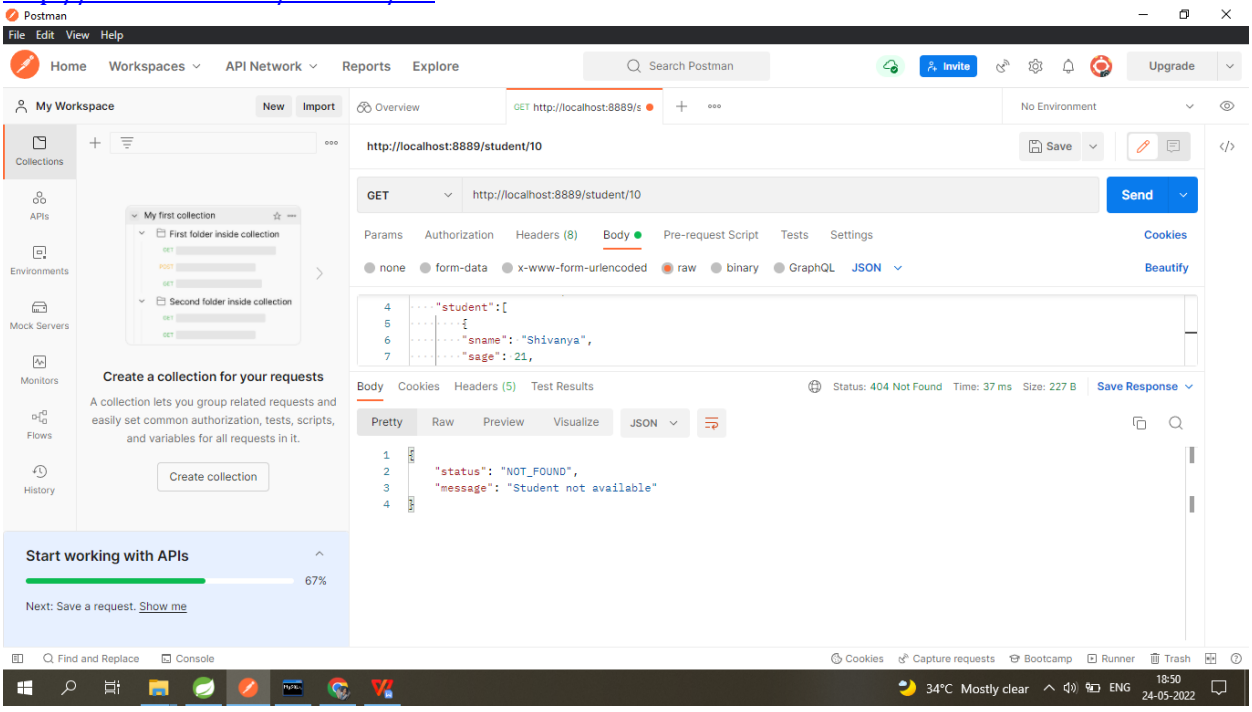
Step 12: Get student record by using student id
http://localhost:8889/student/5/



Step 13: Get student record by using student Name
http://localhost:8889/student/sname/Bhuvana

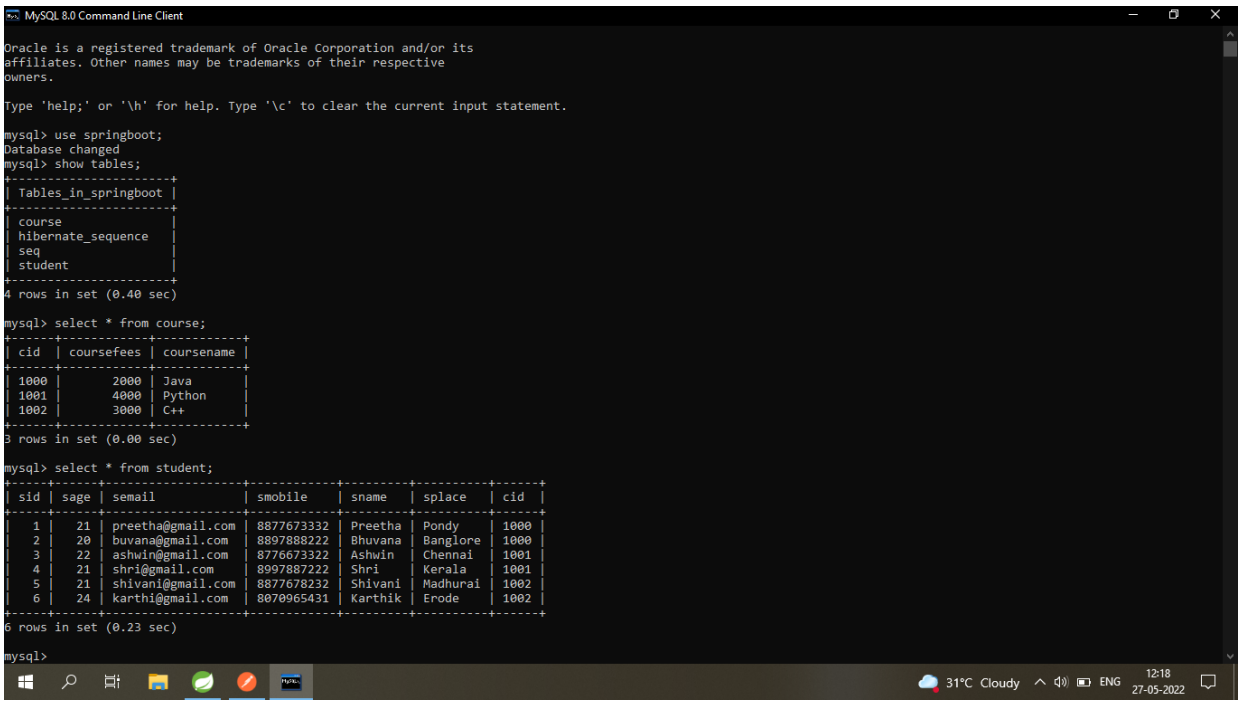


Step 14: Get unknown student detail using student id, it will print error
http://localhost:8889/student/10



9. Database Table Design:

Student Table and course table



10. CONCLUSION:

The Student Management System makes more accessible by giving people an easy way to find and sort information related to it. It allows to view students details as well as course information. The idea is to create a scenario that makes the students to enroll their details into particular courses.