**A PROJECT REPORT ON**

**STUDENT MANAGEMENT SYSTEM**

**Submitted by**

Ms Mohini P. Chavan

Ms. Sakshi S. Kulkarni

Ms. Prajka A. Iratkar

Ms. Vaibhavi Kaware

Ms.Madhavi

**Under the Guidence of**

Trainer Mrs. Indrakka Malli

**Introduction**

In this project we have implemented all students operations like insert, update, delete. In this system we have used two tables Student and course. We used association, exception, security etc.

Student: In student section student can login with their credentials and they can perform the task like add, record, update and delete.

Course: In course section we can perform operations show courses and get all courses.

**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: 2GB

Processor: 64bit

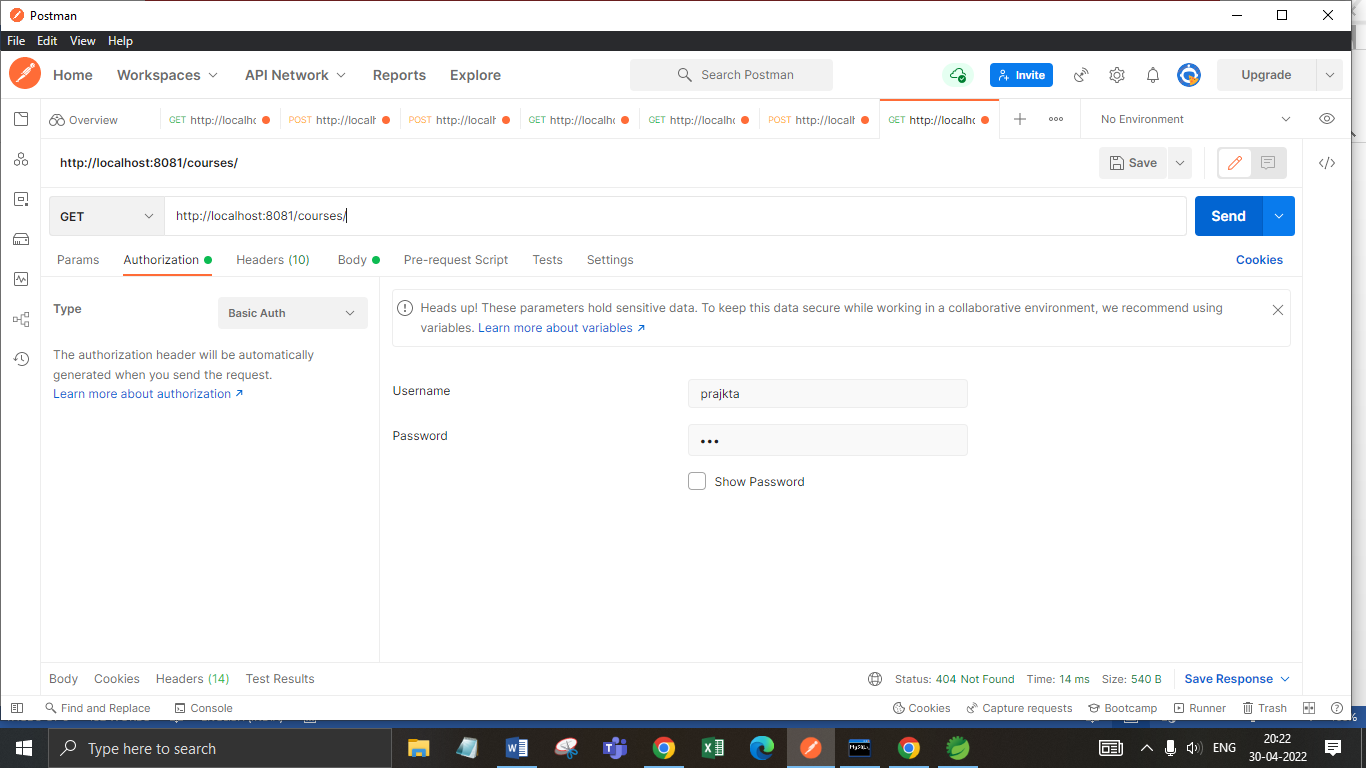
Memory: 512 MB

Disk Space: 100GB

**Screenshot**

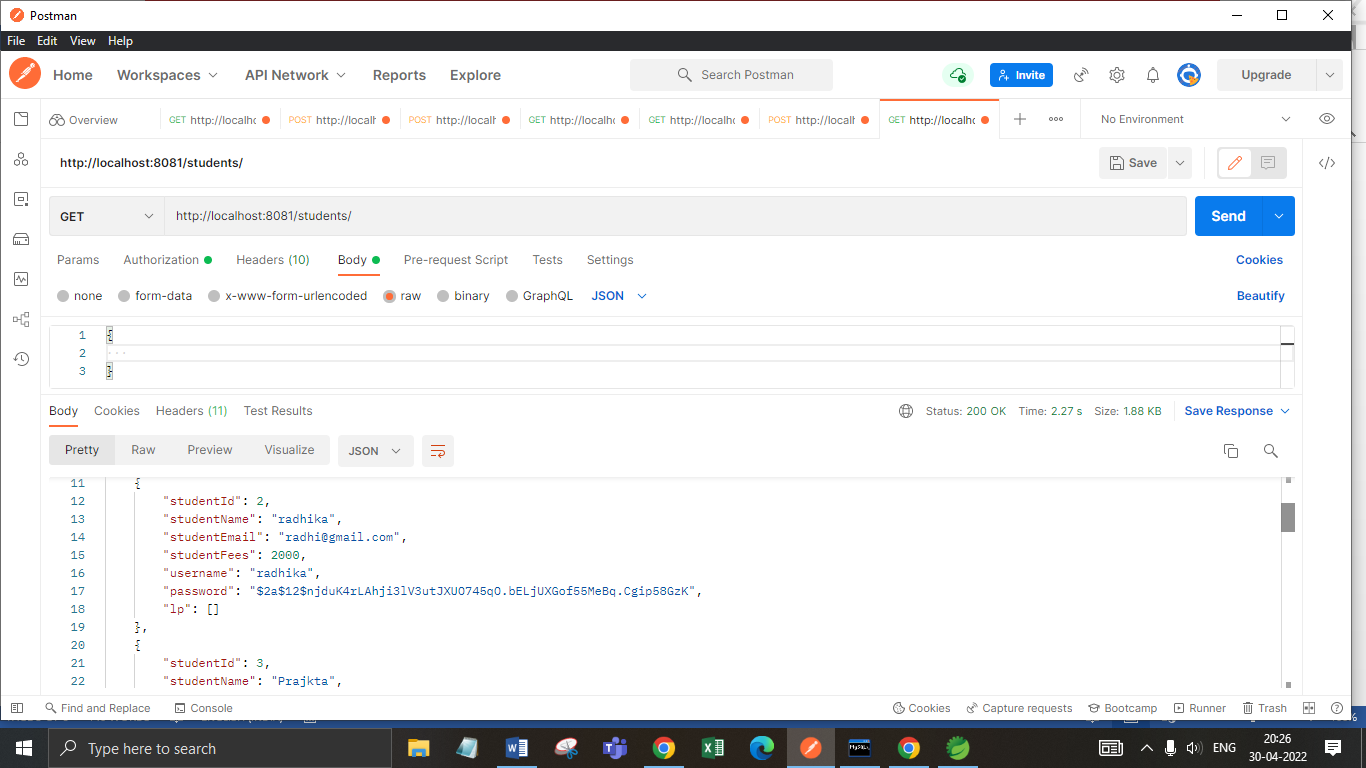
Step 1: Student can login with their username and password.

URL: <http://localhost:8081/students/>



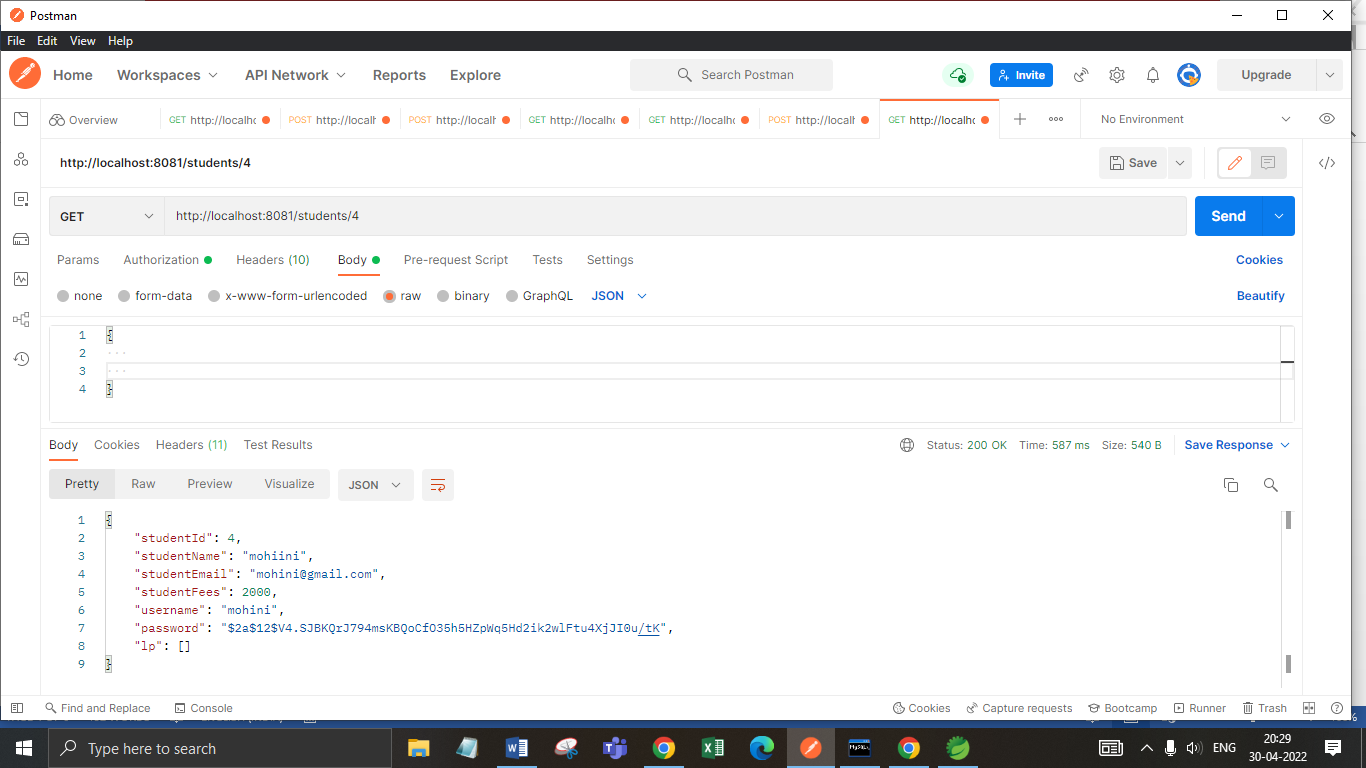
Step 2: Get all students Record.

URL: <http://localhost:8081/students/>



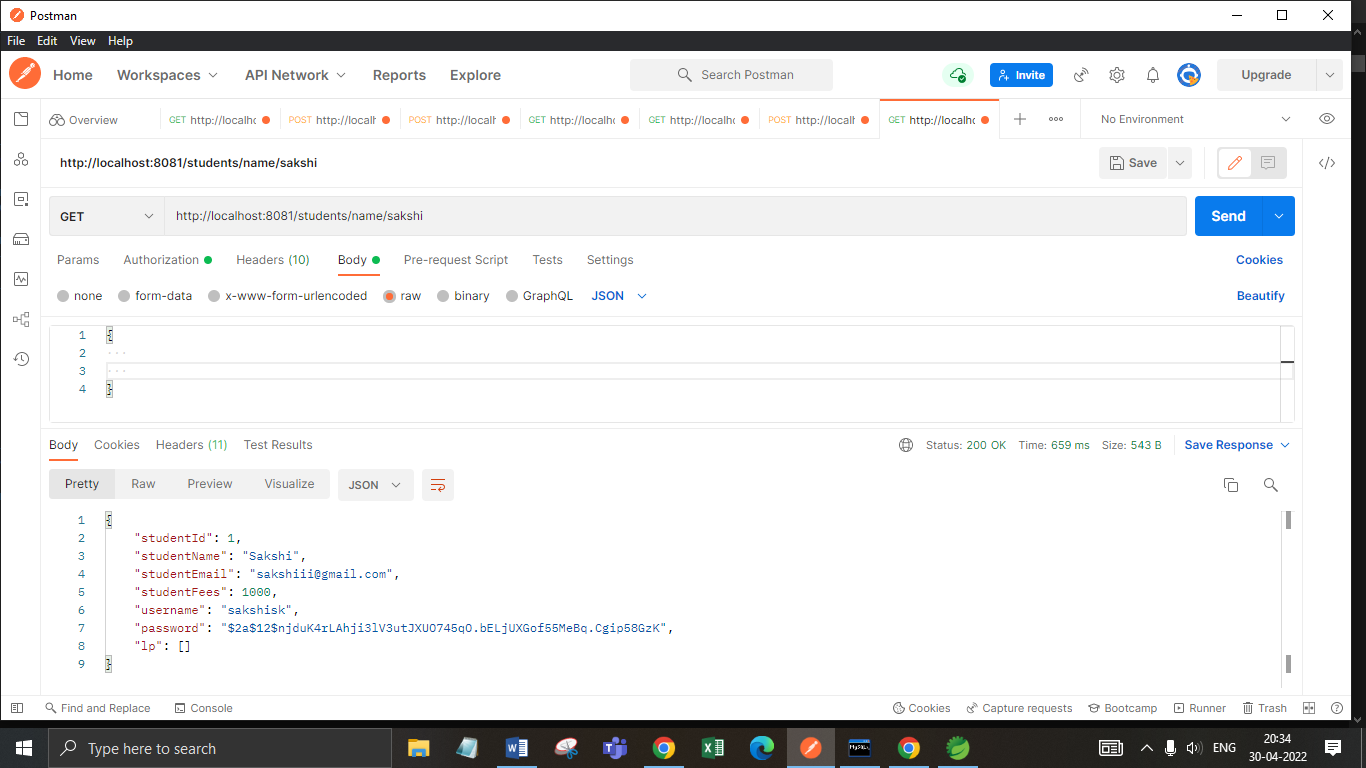
Step 3: Get particular student record by student id.

URL: <http://localhost:8081/students/4>



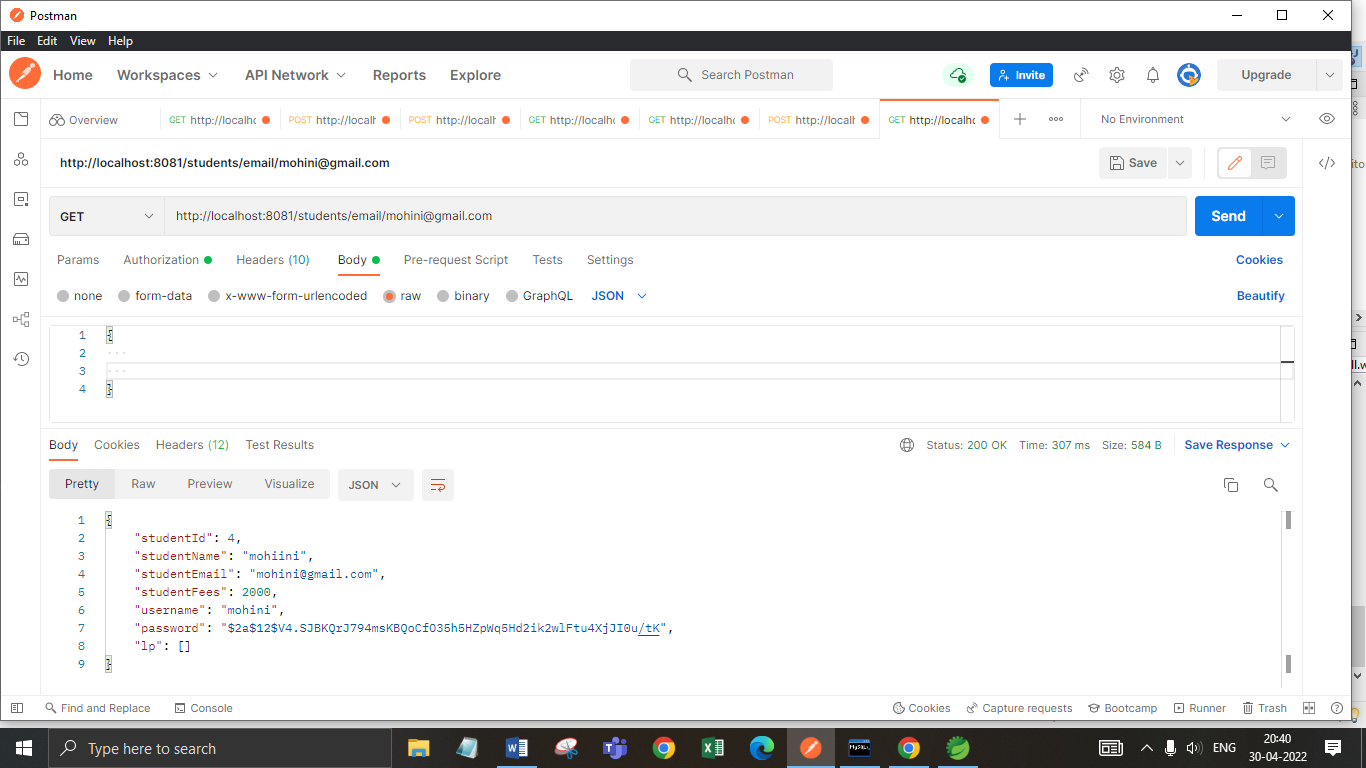
Step 4: Get particular student record by student name.

URL: <http://localhost:8081/students/name/sakshi>



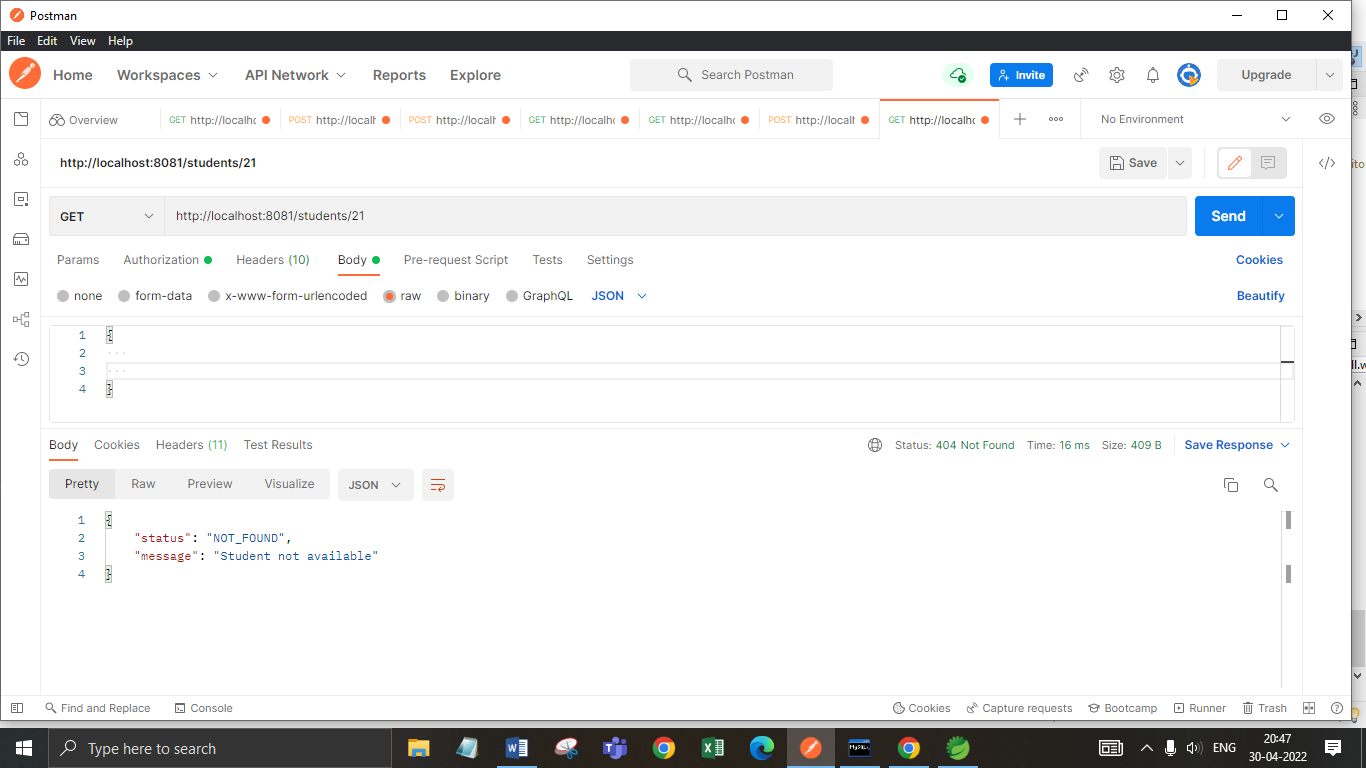
Step 5: Get particular student record by student email.

URL: <http://localhost:8081/students/email/mohini@gmail.com>



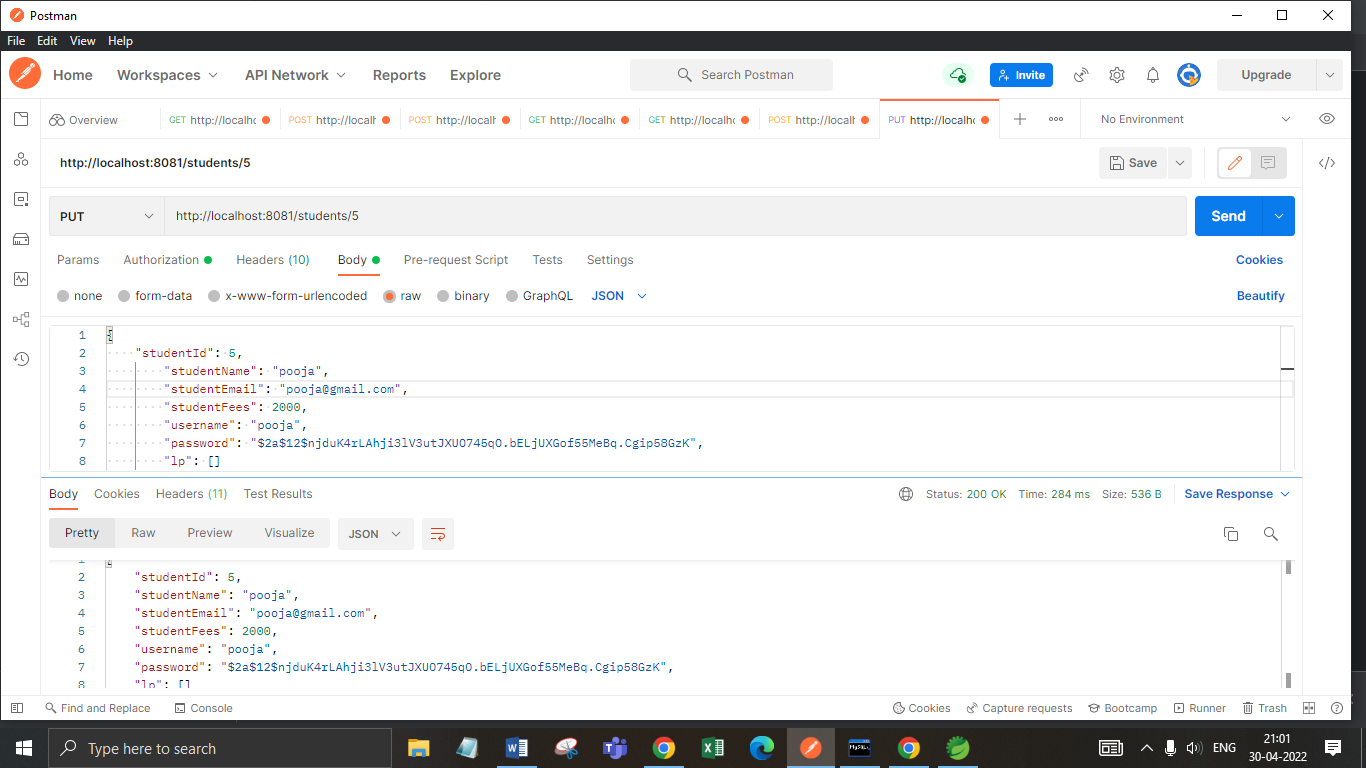
Step 6: If we want to show unavailable student with their id then it shows exception like “student not available”.

URL: <http://localhost:8081/students/21>



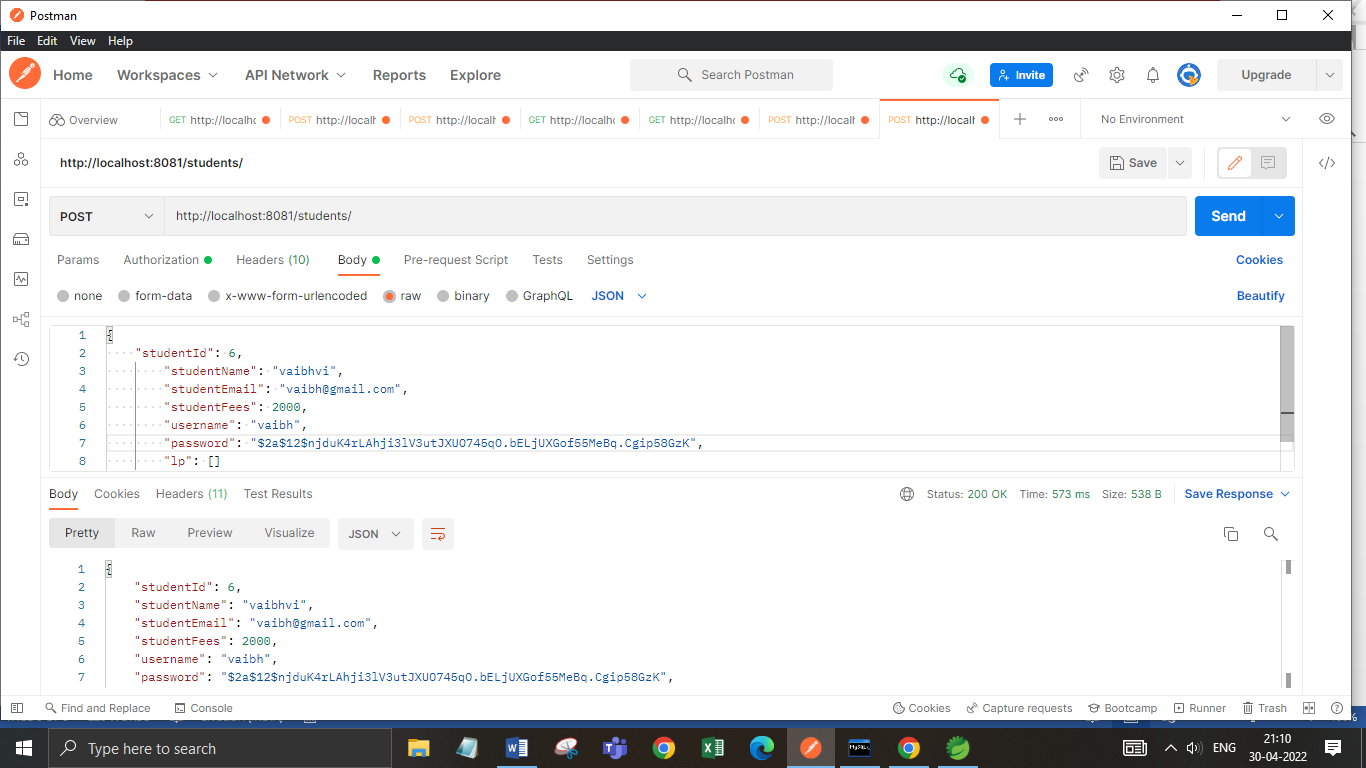
Step 7: We can update student record using put method.

URL: <http://localhost:8081/students/5>



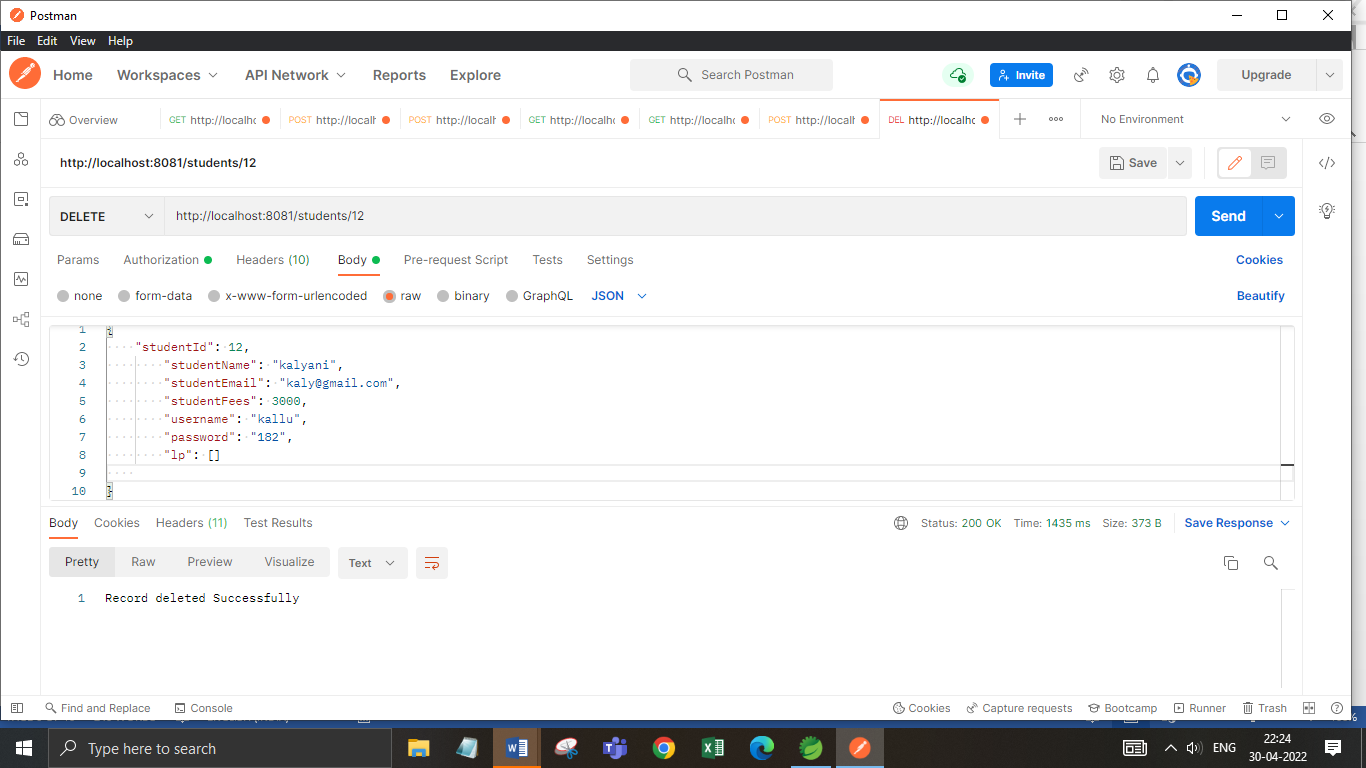
Step 8: We can add student record using post method.

URL: <http://localhost:8081/students/>



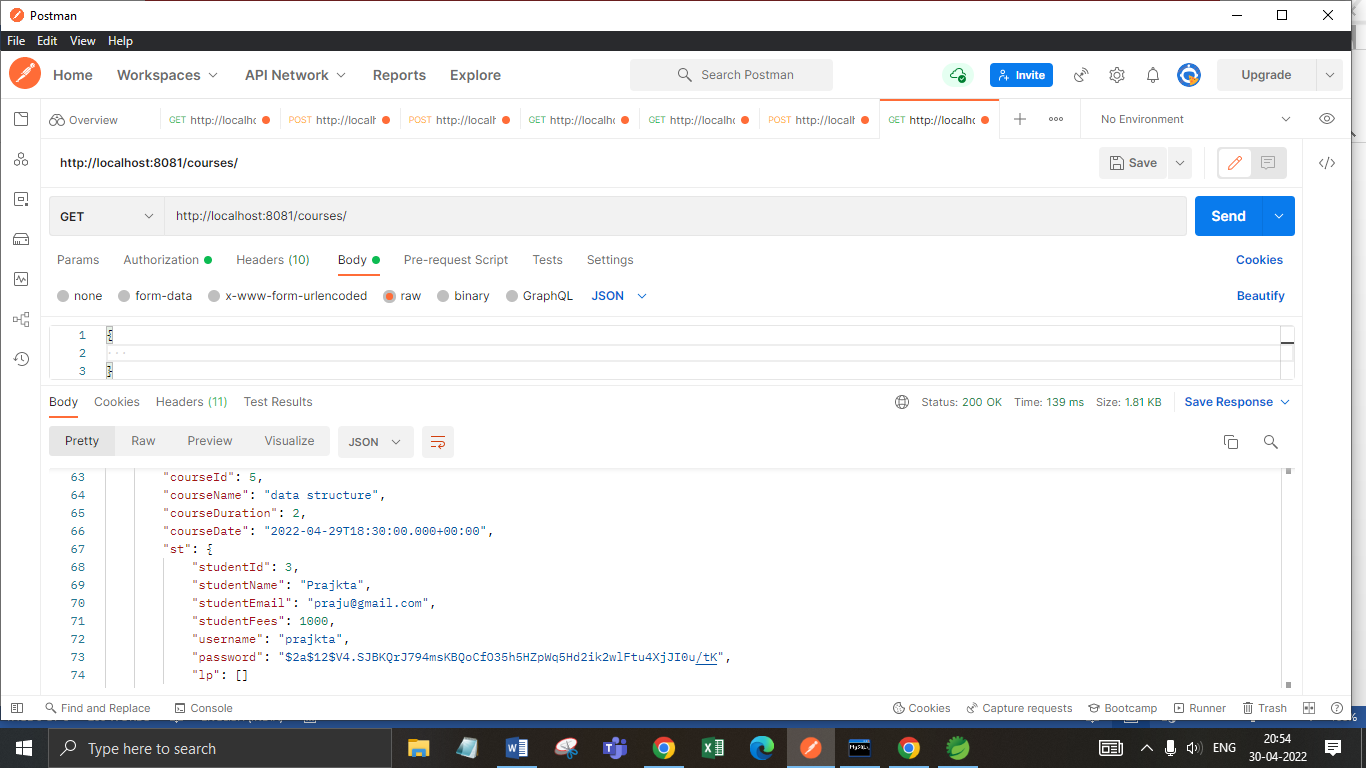
Step 9: We can delete student record using delete method.

URL: <http://localhost:8081/students/12>



Step 10: Get all courses.

URL: <http://localhost:8081/courses/>



**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.

### **@Repository**

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

## **3. @Autowired Annotations**

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](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html" \t "https://howtodoinjava.com/spring5/webmvc/controller-getmapping-postmapping/_blank) 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](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/PostMapping.html" \t "https://howtodoinjava.com/spring5/webmvc/controller-getmapping-postmapping/_blank) 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](https://howtodoinjava.com/spring-core/different-spring-ioc-containers/" \t "https://howtodoinjava.com/spring-core/spring-configuration-annotation/_blank) to generate bean definitions and service requests for those beans at runtime.

### **@OneToMany Annotation**

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](https://www.objectdb.com/api/java/jpa/GeneratedValue) annotation specifies that a value will be automatically generated for that field. This is primarily intended for primary key fields but ObjectDB 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.

**10.@ Exception Handler**

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

## @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.