Title: Building a Student Management System with Spring Boot and MongoDB

Github link: [hacker123shiva/working-with-mongodb: Building a Student Management System with Spring Boot and MongoDB (github.com)]
LinkedId: https://www.linkedin.com/in/shivasrivastava1/

Introduction

In this blog, we'll build a simple Student Management System using Spring Boot and MongoDB. We'll explore CRUD operations, custom queries, error handling, and how to test our API with Postman. Additionally, we'll discuss the use of Lombok for reducing boilerplate code and how to handle exceptions globally.

Project Structure

```
src
└── main
    ├── java
    │   └── com
    │       └── telusko
    │           ├── controller
    │           │   └── StudentController.java
    │           ├── dao
    │           │   └── StudentRepository.java
    │           ├── entity
    │           │   └── Student.java
    │           ├── exception
    │           │   ├── ErrorResponse.java
    │           │   ├── GlobalExceptionHandler.java
    │           │   └── StudentNotFoundException.java
    │           └── service
    │               └── StudentService.java
    └── resources
        └── application.properties
```

1. Setting Up the Spring Boot Application

```java
package com.telusko;

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

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

Explanation:

- `@SpringBootApplication`: This annotation enables auto-configuration and component scanning for the application.

2. Creating the Student Entity

```java
package com.telusko.entity;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

@Data
@Document(collection = "students")
public class Student {
    @Id
    private String id;

    @NotBlank(message = "Name is mandatory")
    private String name;

    @NotNull(message = "Age is required")
```

```
    private Integer age;

    @Email(message = "Email should be valid")
    private String email;
}
```

Explanation of Annotations:

- `@Data`: Lombok generates getters, setters, and other utility methods.
- `@Document`: Marks the class as a MongoDB document and specifies the collection name.
- `@Id`: Marks the field as the primary key.
- Validation annotations (`@NotBlank`, `@NotNull`, `@Email`): Ensure data integrity.

3. Creating the Repository Interface

```
package com.telusko.dao;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import com.telusko.entity.Student;
import java.util.List;

@Repository
public interface StudentRepository extends MongoRepository<Student, String>
{
    List<Student> findByName(String name);
    List<Student> findByAgeGreaterThan(int age);
}
```

Explanation:

- `MongoRepository`: Provides CRUD operations and query methods.
- Custom query methods: Automatically implemented by Spring Data based on method names.

4. Implementing the Service Layer

```
package com.telusko.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.telusko.dao.StudentRepository;
import com.telusko.entity.Student;
import com.telusko.exception.StudentNotFoundException;
import java.util.List;

@Service
public class StudentService {
    @Autowired
    private StudentRepository studentRepository;

    public Student createStudent(Student student) {
        return studentRepository.save(student);
    }

    public List<Student> getAllStudents() {
        return studentRepository.findAll();
    }

    public Student getStudentById(String id) {
        return studentRepository.findById(id)
            .orElseThrow(() -> new StudentNotFoundException("Student not
found with id: " + id));
    }

    public Student updateStudent(String id, Student student) {
        if (!studentRepository.existsById(id)) {
            throw new StudentNotFoundException("Student not found with id:
" + id);
        }
        student.setId(id);
        return studentRepository.save(student);
    }

    public void deleteStudent(String id) {
        if (!studentRepository.existsById(id)) {
            throw new StudentNotFoundException("Student not found with id:
```

```
" + id);
        }
        studentRepository.deleteById(id);
    }

    public List<Student> findByName(String name) {
        return studentRepository.findByName(name);
    }

    public List<Student> findByAgeGreaterThan(int age) {
        return studentRepository.findByAgeGreaterThan(age);
    }
}
```

Explanation:

- **CRUD operations**: Basic operations for creating, reading, updating, and deleting students.
- **Custom Queries**: Methods for querying based on `name` and `age`.

5. Creating the Controller

```
package com.telusko.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.telusko.entity.Student;
import com.telusko.service.StudentService;
import jakarta.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/students")
public class StudentController {
    @Autowired
    private StudentService studentService;

    @PostMapping
    public Student createStudent(@Valid @RequestBody Student student) {
        return studentService.createStudent(student);
```

```
    }

    @GetMapping
    public List<Student> getAllStudents() {
        return studentService.getAllStudents();
    }

    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable String id) {
        return studentService.getStudentById(id);
    }

    @PutMapping("/{id}")
    public Student updateStudent(@PathVariable String id, @Valid
@RequestBody Student student) {
        return studentService.updateStudent(id, student);
    }

    @DeleteMapping("/{id}")
    public void deleteStudent(@PathVariable String id) {
        studentService.deleteStudent(id);
    }

    @GetMapping("/search/byName")
    public List<Student> findByName(@RequestParam String name) {
        return studentService.findByName(name);
    }

    @GetMapping("/search/byAgeGreaterThan")
    public List<Student> findByAgeGreaterThan(@RequestParam int age) {
        return studentService.findByAgeGreaterThan(age);
    }
}
```

6. Handling Exceptions

ErrorResponse Entity

```
package com.telusko.exception;
import lombok.AllArgsConstructor;
import lombok.Data;
@Data
```

```
@AllArgsConstructor
public class ErrorResponse {
   private String message;
   private String details;
}
```

Global Exception Handler

```
package com.telusko.exception;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class ErrorResponse {
    private String message;
    private String details;
}

@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(StudentNotFoundException.class)
    public ResponseEntity<ErrorResponse>
handleStudentNotFoundException(StudentNotFoundException ex, WebRequest
request) {
        ErrorResponse errorResponse = new ErrorResponse(ex.getMessage(),
request.getDescription(false));
        return new ResponseEntity<>(errorResponse, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleGlobalException(Exception
ex, WebRequest request) {
        ErrorResponse errorResponse = new ErrorResponse("Internal Server
Error", request.getDescription(false));
        return new ResponseEntity<>(errorResponse,
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Explanation:

- `@ControllerAdvice`: Handles exceptions globally.
- Custom exceptions throw meaningful messages that are returned as part of the response.

Custom Exception Class

```
package com.telusko.exception;
public class StudentNotFoundException extends RuntimeException {

      private static final long serialVersionUID = 1L;
      public StudentNotFoundException(String message) {
       super(message);
   }
}
```

7. Application Configuration

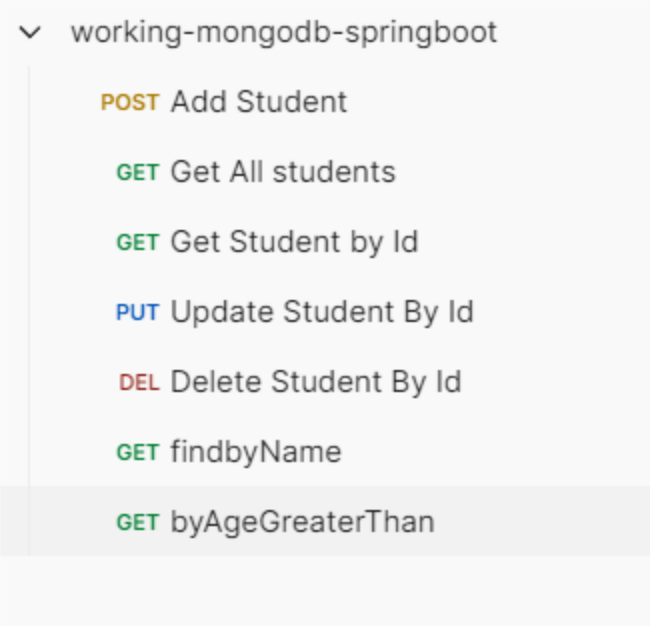
```
spring.application.name=working-mongodb
spring.data.mongodb.uri=mongodb://localhost:27017/school
spring.data.mongodb.database=school
logging.level.org.springframework.data.mongodb.core.MongoTemplate=DEBUG
```

8. Testing with Postman

Starting with mongosh Shell

1. Install MongoDB and ensure it's running.
2. Open a terminal and start `mongosh`.

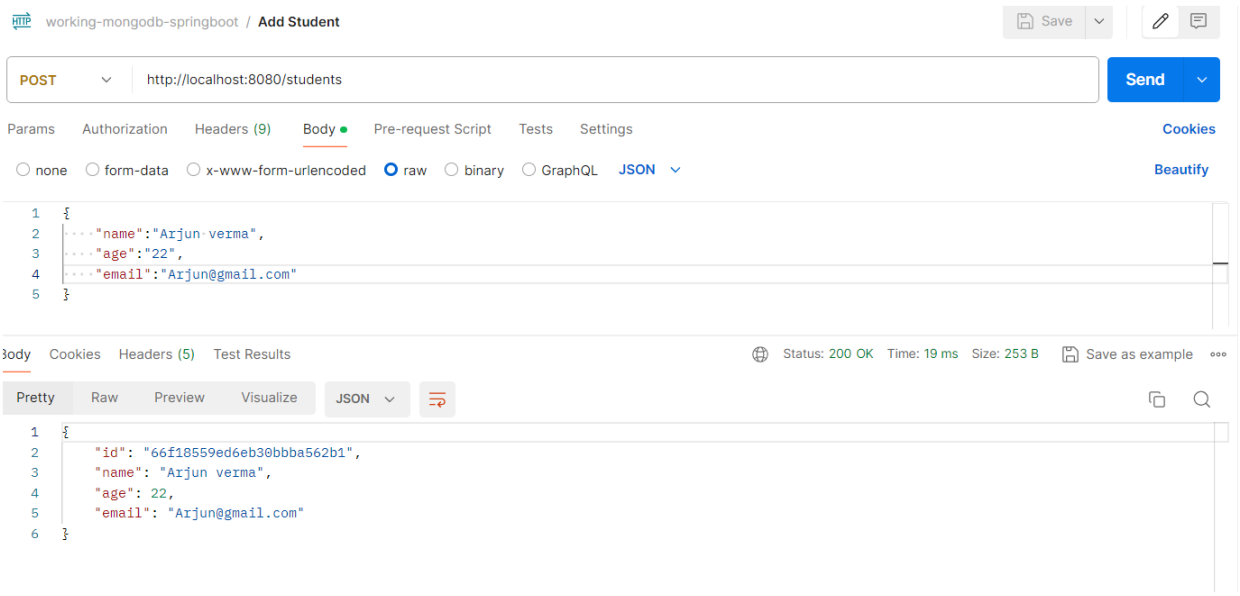
```
use school
```



1. Create a Student

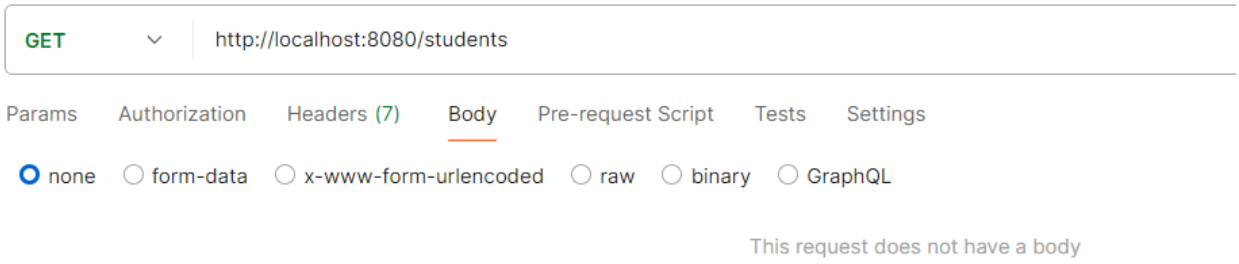
- **Method**: POST
- **URL**: `http://localhost:8080/students`
- **Body**:

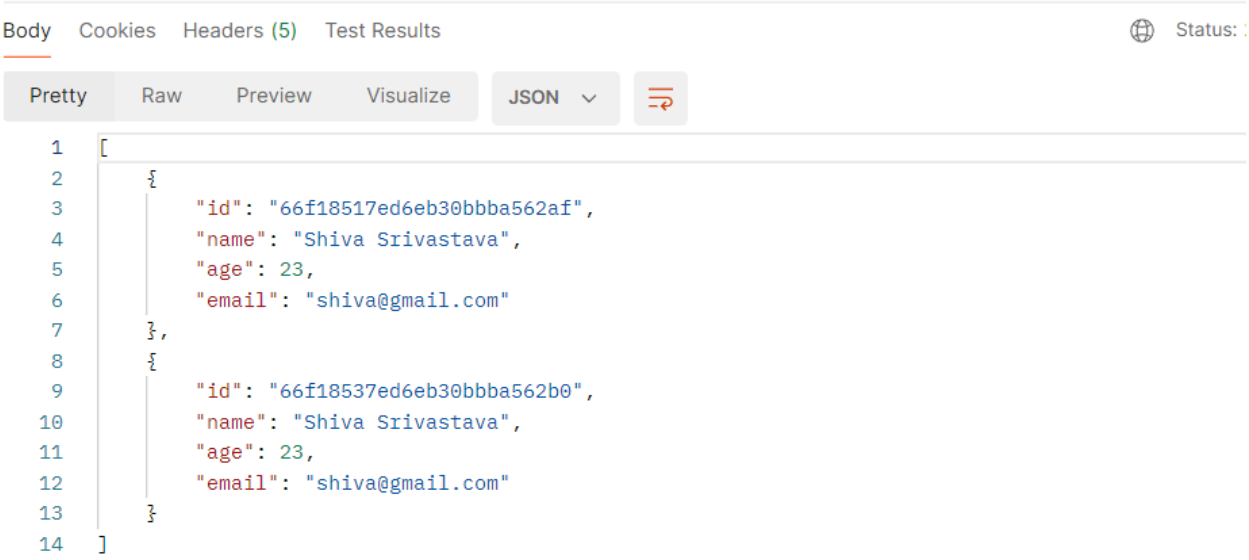
```
{
    "name": "Shiva Srivastava",
    "age": 23,
    "email": "shiva@gmail.com"
}
```



2. Get All Students

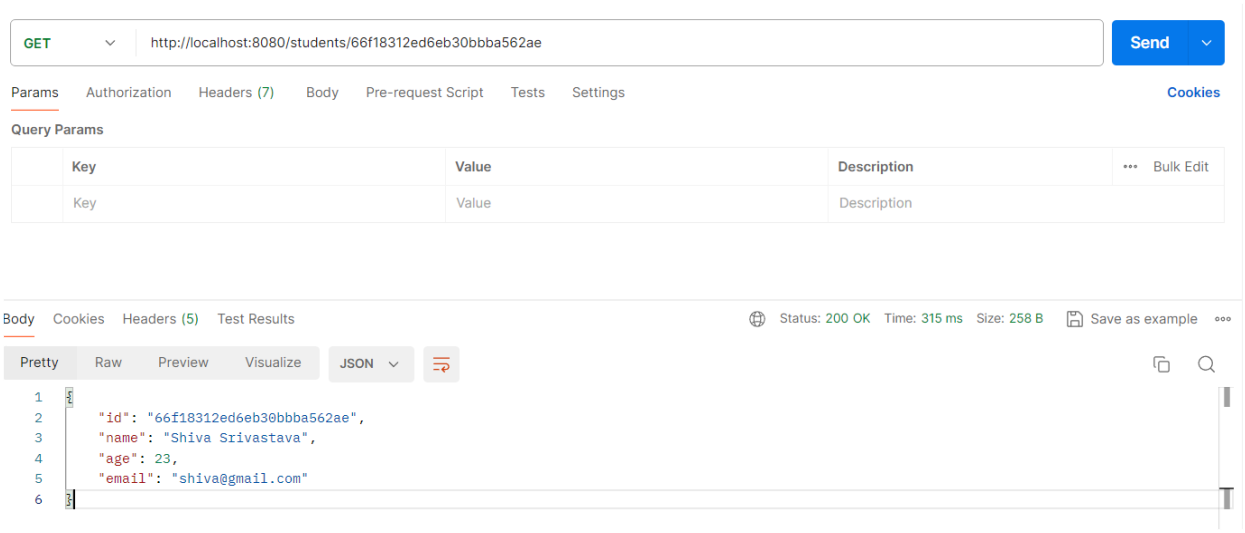
- **Method**: GET
- **URL**: `http://localhost:8080/students`





3. Get Student by ID

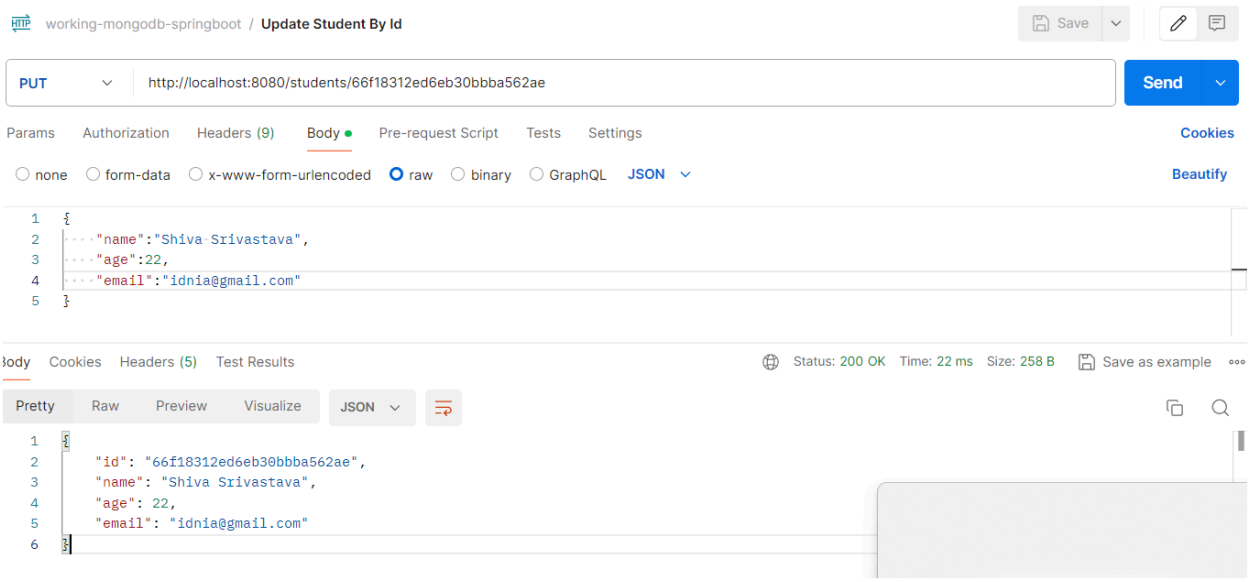
- **Method**: GET
- **URL**: `http://localhost:8080/students/{id}`



4. Update a Student

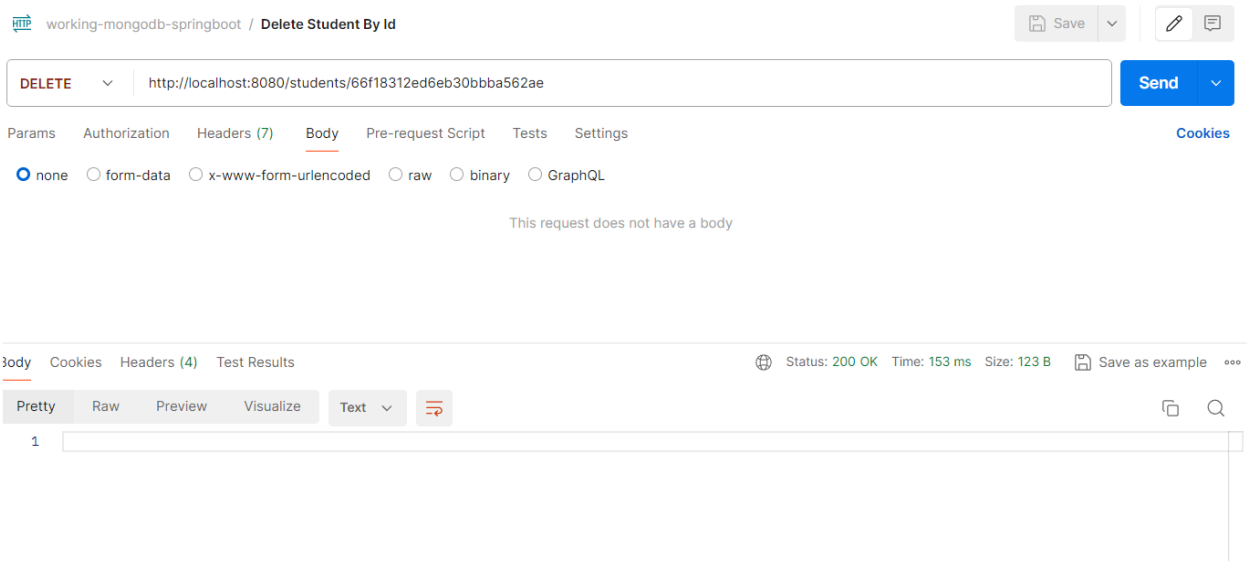
- **Method**: PUT
- **URL**: `http://localhost:8080/students/{id}`
- **Body**:

```
{
    "name": "Shiva Srivastava",
    "age": 23,
    "email": "ind@gmail.com"
}
```



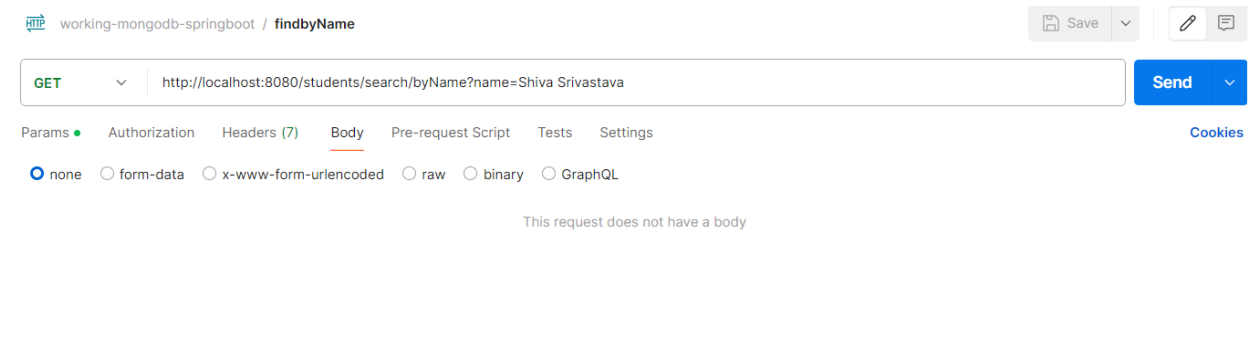
5. Delete a Student

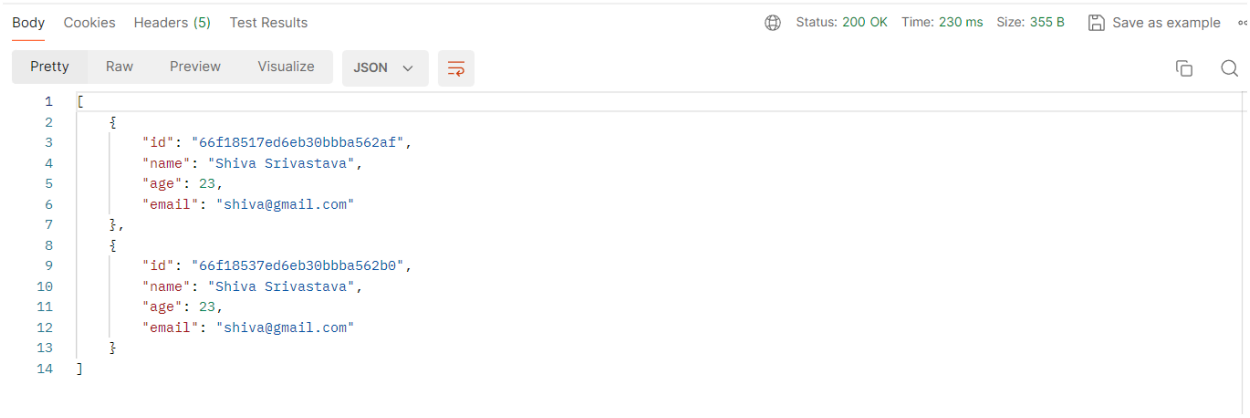
- **Method**: DELETE
- **URL**: `http://localhost:8080/students/{id}`



6. Search by Name

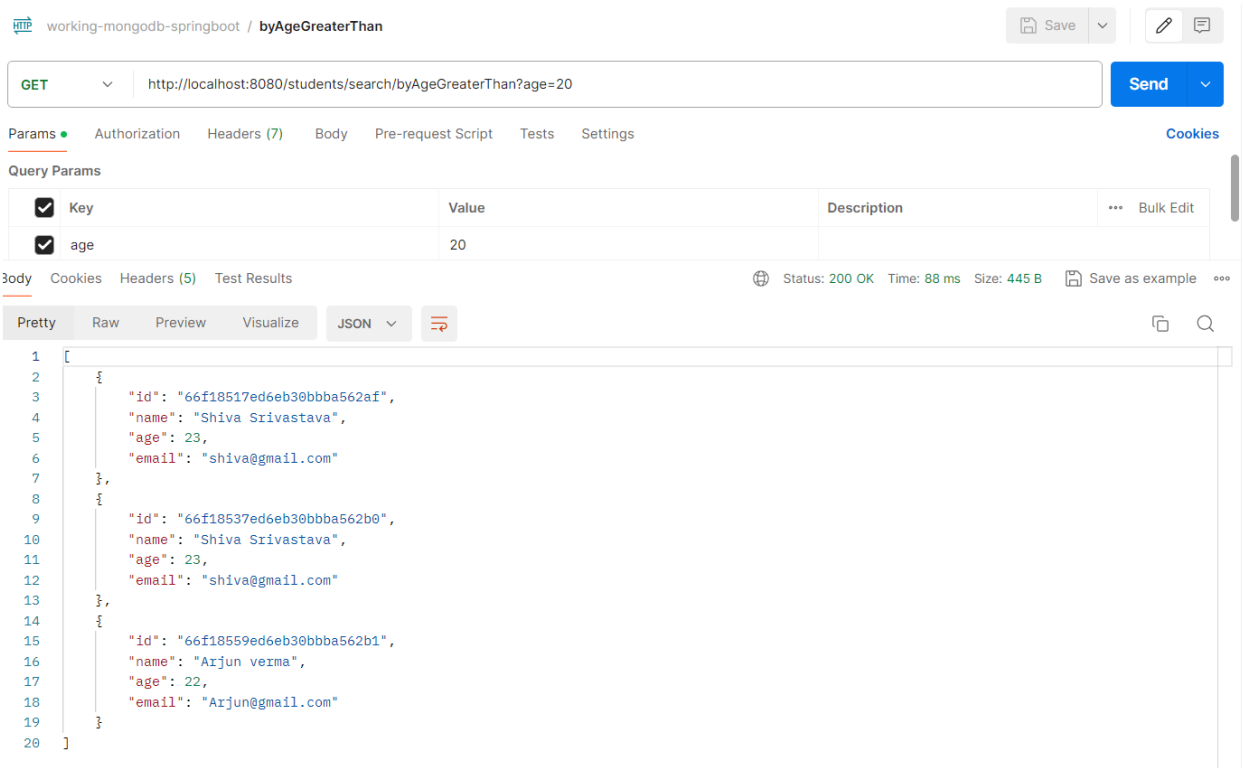
- **Method**: GET
- **URL**: `http://localhost:8080/students/search/byName?name=John`





7. Search by Age Greater Than

- **Method**: GET
- **URL**: `http://localhost:8080/students/search/byAgeGreaterThan?age=18`


working-mongodb-springboot / byAgeGreaterThan
Save
GET
http://localhost:8080/students/search/byAgeGreaterThan?age=20
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
age
20
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 88 ms
Size: 445 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": "66f18517ed6eb30bbba562af",
"name": "Shiva Srivastava",
"age": 23,
"email": "shiva@gmail.com"
},
{
"id": "66f18537ed6eb30bbba562b0",
"name": "Shiva Srivastava",
"age": 23,
"email": "shiva@gmail.com"
},
{
"id": "66f18559ed6eb30bbba562b1",
"name": "Arjun verma",
"age": 22,
"email": "Arjun@gmail.com"
}
]