

# TASK1

## JAVA REST API EXAMPLE

### **Student Information System Using Spring Boot and MongoDB**

**Name:** Nishtha Singh (19BCE0037)

**Email:** [nishtha.singh2019@vitstudent.ac.in](mailto:nishtha.singh2019@vitstudent.ac.in)

**College:** VIT VELLORE

## **About The Project**

The project involves building a REST service using Spring Boot and MongoDB, which allows users to manage student information through a set of API endpoints. The service is designed to support various CRUD operations on student data. The student's collection in the MongoDB database contains information about each student's name, student number, email, course list, and GPA.

The project follows a structured approach, beginning with setting up the development environment by installing the required tools such as JDK, IntelliJ, Maven, MongoDB, and Postman. The project uses the Spring Initializer to set up a basic Spring Boot project and add necessary dependencies.

The next step involves defining the model for the student object, creating a repository interface to communicate with the MongoDB database, and implementing a service layer to handle business logic.

The service layer includes methods for retrieving student information by student number or email, fetching all students, adding new students to the system, updating student information, and deleting students

from the system. The project also includes data validation and error handling, caching and performance optimization, security measures such as authentication and authorization, and logging and monitoring.

To test the REST service, the project uses tools such as Postman, which allows users to send HTTP requests to the API endpoints and receive responses. In summary, the project demonstrates how to build a functional REST service using Spring Boot and MongoDB, emphasizing simplicity and a structured approach to implementing the business logic.

## **Steps**

1. Install JDK 8.0, IntelliJ, Maven, MongoDB, and Postman.
2. Use Spring Initializer to create a new Spring Boot project.
3. Create a model and repository for the "Student" class.
4. Create a StudentService that implements 6 methods for CRUD operations.
5. Now, the student service will be called by student controller.  
Simply, these methods list students according to some criteria, save a student, update a student and delete a student.
6. Create a REST controller with @GetMapping, @PostMapping, and @DeleteMapping annotations.
7. Next is Database configuration for connecting to Mongo dB database
8. To test the REST service in a Java project, a tool like Postman is used to send HTTP requests and receive responses.

# Results

## GET REQUESTS

### 404 – Server Doesn't exist as URL is wrong

The screenshot shows a REST client interface with a GET request to `localhost:8081/api/students/`. The request body is a JSON object representing a student: `{ "name": "riya", "studentNumber": "24", "email": "riya12@gmail.com", "courseList": [ "English", "Science" ], "gpa": 7.55 }`. The response is a 404 Not Found error, indicating the resource does not exist. The response body is a JSON object: `{ "timestamp": "2023-03-23T21:50:53.797+00:00", "status": 404, "error": "Not Found", "path": "/api/students/" }`.

```
GET localhost:8081/api/students/

{"name": "riya", "studentNumber": "24", "email": "riya12@gmail.com", "courseList": [ "English", "Science" ], "gpa": 7.55}

404 Not Found 22 ms 362 B Save Response

{"timestamp": "2023-03-23T21:50:53.797+00:00", "status": 404, "error": "Not Found", "path": "/api/students/"}
```

### 500 – Internal Server Error

GET localhost:8081/student + ... No Environment

localhost:8081/students/ Save

GET localhost:8081/students/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (4) Test Results 500 Internal Server Error 30.42 s 264 B Save Response

## 200 ok – Empty Student List

GET localhost:8081/student + ... No Environment

localhost:8081/students/ Save

GET localhost:8081/students/ Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results 200 OK 362 ms 166 B Save as Example

Pretty Raw Preview Visualize JSON

1

## 200 ok – Order students by their GPA

The screenshot shows a REST client interface with the following details:

- Request:** GET localhost:8081/students/orderByGpa
- Response Status:** 200 OK, 122 ms, 596 B
- Response Body (JSON):**

```
[
  {
    "id": "641d766e8c62da02e8a5cc5d",
    "name": "devansh",
    "studentNumber": 30,
    "email": "devansh22@gmail.com",
    "courseList": [
      "English",
      "Maths",
      "Science"
    ],
    "gpa": 8.23
  },
  {
    "id": "641d74078c62da02e8a5cc5c",
    "name": "riya",
    "studentNumber": 28,
    "email": "riya28@gmail.com",
    "courseList": [
      "Hindi",
      "Maths"
    ],
    "gpa": 7.44
  },
  {
    "id": "641d76c28c62da02e8a5cc5e"
  }
]
```

The interface includes tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings. The Body tab is active, showing the JSON response in Pretty format. The status bar at the bottom includes Cookies, Capture requests, Runner, Trash, and other utility icons.

# POST REQUESTS

## 200 ok – Adding a student

The screenshot shows a REST client interface with a POST request to `localhost:8081/students/save`. The request body is a JSON object:

```
1 {
2   "name": "riya",
3   "studentNumber": 28,
4   "email": "riya28@gmail.com",
5   "courseList": ["Hindi", "Maths"],
6   "gpa": 7.44
7 }
```

The response status is **200 OK** with a response time of **465 ms** and a body size of **190 B**. The response body is:

```
1 Student added successfully
```

The screenshot shows a REST client interface with a POST request to `localhost:8081/students/save`. The request body is a JSON object:

```
1 {
2   "name": "sara",
3   "studentNumber": 42,
4   "email": "saramalik@gmail.com",
5   "courseList": ["Maths", "Science"],
6   "gpa": 6.36
7 }
```

The response status is **200 OK** with a response time of **19 ms** and a body size of **190 B**. The response body is:

```
1 Student added successfully
```



# DELETE REQUESTS

## 200 ok – Deleting student by their studentNumber

The screenshot shows a REST client interface with a DELETE request to `localhost:8081/students/delete/42`. The response is `200 OK` with a body message `Student deleted successfully`.

Request details:

- Method: DELETE
- URL: localhost:8081/students/delete/42

Response details:

- Status: 200 OK
- Time: 71 ms
- Size: 192 B
- Body: Student deleted successfully

This is a close-up view of the response body, showing the message `Student deleted successfully`.

# PUT REQUESTS

## 200 ok – Updating student's GPA by their studentNumber

The screenshot displays a REST client interface with the following details:

- Request Method:** PUT
- URL:** localhost:8081/students/28/updateGpa/9.40
- Status:** 200 OK
- Response Time:** 610 ms
- Response Size:** 342 B
- Response Body (JSON):**

```
{  "id": "641d74078c62da02e8a5cc5c",  "name": "riya",  "studentNumber": 28,  "email": "riya28@gmail.com",  "courseList": [    "Hindi",    "Maths"  ],  "gpa": 9.4}
```

The interface also shows tabs for Params, Authorization, Headers (9), Body, Pre-request Script, Tests, and Settings. The Body tab is currently selected, showing the JSON response in a pretty-printed format.

# THE STUDENTS COLLECTION

The screenshot shows a MongoDB terminal window with the following content:

```
apple -- mongod --dbpath /usr/local/mongodb -- 80x24
{
  "client": "conn7",
  "doc": {
    "driver": {
      "name": "mongo",
      "version": "4.6.13",
      "os": {
        "type": "Darwin",
        "name": "Mac OS X",
        "version": "11.6.4",
        "platform": "Java/Homebrew"
      },
      "platform": "Java/Homebrew"
    },
    "context": {
      "conn": "conn7",
      "msg": "createCollection",
      "uid": {
        "uid": "71a074e2-379ad62c840f",
        "options": {}
      },
      "date": "2023-03-24T15:27:27.298+05:30"
    },
    "collection": {
      "name": "test.students",
      "index": {
        "id": "id",
        "indexId": "collection-0-100645424188123",
        "collection": "collection-0-100645424188123",
        "date": "2023-03-24T15:27:27.298+05:30"
      },
      "context": {
        "conn": "conn7",
        "msg": "Index build: done",
        "uid": {
        "uid": "71a074e2-379ad62c840f",
        "options": {}
      },
      "date": "2023-03-24T15:27:27.298+05:30"
    },
    "command": {
      "insert": "students",
      "orders": {
        "id": "id",
        "indexId": "collection-0-100645424188123",
        "collection": "collection-0-100645424188123",
        "date": "2023-03-24T15:27:27.298+05:30"
      },
      "context": {
        "conn": "conn7",
        "msg": "Slow query",
        "uid": {
        "uid": "71a074e2-379ad62c840f",
        "options": {}
      },
      "date": "2023-03-24T15:27:27.298+05:30"
    },
    "featureCompatibilityVersion": {
      "major": 3,
      "minor": 0,
      "patch": 0,
      "build": 0,
      "compatibility": "Global",
      "acquireCount": {
        "w": 3,
        "r": 0
      },
      "flowControl": {
        "storage": {
          "id": "id",
          "indexId": "collection-0-100645424188123",
          "collection": "collection-0-100645424188123",
          "date": "2023-03-24T15:27:27.298+05:30"
        },
        "remote": "127.0.0.1:49942",
        "pro"
      }
    }
  }
}
```

Below the terminal window, there is a table with 4 columns and 17 rows of data:

requests and easy set common	4
authorization, tests, scripts, and	5
variables for all requests in it.	6
	7
	8
	9
	10
	11
	12
	13
	14
	15
	16
	17

On the right side of the terminal window, there is a JSON document representing a student record:

```
{
  "_id": "641d74078c62da02e8a5cc5c*",
  "name": "riya",
  "studentNumber": NumberLong(28),
  "email": "riya28@gmail.com",
  "courseList": [
    "Hindi",
    "Maths"
  ],
  "gpa": 7.440000057228459,
  "_class": "com.student.infosystem.studentinfosystem.model.Student"
}
```

Below the JSON document, there is another JSON document representing a student record:

```
{
  "_id": "641d766e8c62da02e8a5cc5d*",
  "name": "devansh",
  "studentNumber": NumberLong(30),
  "email": "devansh22@gmail.com",
  "courseList": [
    "English",
    "Maths",
    "Science"
  ],
  "gpa": 8.2299999542236328,
  "_class": "com.student.infosystem.studentinfosystem.model.Student"
}
```

At the bottom of the terminal window, there is a third JSON document representing a student record:

```
{
  "_id": "641d780b8c62da02e8a5cc5f*",
  "name": "ajay",
  "studentNumber": NumberLong(45),
  "email": "ajaysingh@gmail.com",
  "courseList": [
    "English",
    "Maths",
    "Hindi"
  ],
  "gpa": 8.0500000190734863,
  "_class": "com.student.infosystem.studentinfosystem.model.Student"
}
```

```
apple — mongo — 96x41
---
> db.students.find().pretty()
{
  "_id" : ObjectId("641d74078c62da02e8a5cc5c"),
  "name" : "riya",
  "studentNumber" : NumberLong(28),
  "email" : "riya28@gmail.com",
  "courseList" : [
    "Hindi",
    "Maths"
  ],
  "gpa" : 7.440000057220459,
  "_class" : "com.student.infosystem.studentinfosystem.model.Student"
}
{
  "_id" : ObjectId("641d766e8c62da02e8a5cc5d"),
  "name" : "devansh",
  "studentNumber" : NumberLong(30),
  "email" : "devansh22@gmail.com",
  "courseList" : [
    "English",
    "Maths",
    "Science"
  ],
  "gpa" : 8.229999542236328,
  "_class" : "com.student.infosystem.studentinfosystem.model.Student"
}
{
  "_id" : ObjectId("641d780b8c62da02e8a5cc5f"),
  "name" : "ajay",
  "studentNumber" : NumberLong(45),
  "email" : "ajaysingh@gmail.com",
  "courseList" : [
    "English",
    "Maths",
    "Hindi"
  ],
  "gpa" : 8.050000190734863,
  "_class" : "com.student.infosystem.studentinfosystem.model.Student"
}
>
```