Laboratory Record
of : EAD

Roll No.:
Exp No.: _____
Sheet No.: _____
Date: _____

Aim:Build a Rest API for student CRUD operations

1) Prerequisites

- Node.js 18+ installed. Verify: node -v (and npm -v)

- A code editor (VS Code recommended).

- A MongoDB database: either a free MongoDB Atlas cluster or a local MongoDB instance.

- (Optional) API client like Postman or VS Code Thunder Client to test endpoints.

2) Create Project Folder

Open your terminal and run:

```
mkdir student-crud-api
cd student-crud-api
npm init -y
```

3) Install Dependencies

We will use Express for the HTTP server, Mongoose for MongoDB, CORS to allow front-end calls, dotenv for environment variables, and nodemon for auto-restart during development.

```
npm i express mongoose cors dotenv
npm i -D nodemon
```

4) Project Structure

```
student-crud-api/
  package.json
  .env              # will be created by you
  src/
    server.js
    db.js
    models/
      Student.js
```

```
routes/
  students.js
controllers/
  studentsController.js
```

5) Configure package.json Scripts

Open package.json and add scripts:

```
{
  "name": "student-crud-api",
  "version": "1.0.0",
  "main": "src/server.js",
  "type": "module",
  "scripts": {
    "dev": "nodemon src/server.js",
    "start": "node src/server.js"
  },
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "mongoose": "^8.6.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

6) Create .env

Create a file named .env in the project root with:

```
PORT=3000
MONGODB_URI=mongodb://127.0.0.1:27017/studentdb  # or your MongoDB Atlas connection string
```

If you use MongoDB Atlas, replace the URI with your connection string, e.g.:

MONGODB_URI=mongodb+srv://<USERNAME>:<PASSWORD>@<CLUSTER>.mongodb.net/studentdb?retryWrites=true&w=majority

7) Database Connection (src/db.js)

```
import mongoose from "mongoose";

export async function connectDB(uri) {
  try {
    mongoose.set("strictQuery", true);
    await mongoose.connect(uri, { dbName: "studentdb" });
    console.log("✅ Connected to MongoDB");
  } catch (err) {
    console.error("❌ MongoDB connection error:", err.message);
    process.exit(1);
  }
}
```

8) Student Model (src/models/Student.js)

Define the schema fields and basic validation.

```
import mongoose from "mongoose";

const studentSchema = new mongoose.Schema(
  {
    name: { type: String, required: true, trim: true },
    rollNo: { type: String, required: true, unique: true, trim: true },
    branch: { type: String, required: true, enum: ["IT", "CSE", "ECE", "EEE", "ME", "CE"], },
    year: { type: Number, required: true, min: 1, max: 4 },
    email: { type: String, required: true, lowercase: true, match: /.+@.+\..+/ },
  },
  { timestamps: true }
);

export const Student = mongoose.model("Student", studentSchema);
```

9) Controller (src/controllers/studentsController.js)

```
import { Student } from "../models/Student.js";

export async function createStudent(req, res) {
  try {
    const student = await Student.create(req.body);
    res.status(201).json(student);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
}

export async function getStudents(req, res) {
  try {
    const { page = 1, limit = 10, q = "" } = req.query;
    const filter = q ? { $or: [
      { name: { $regex: q, $options: "i" } },
      { rollNo: { $regex: q, $options: "i" } },
      { branch: { $regex: q, $options: "i" } }
    ] } : {};

    const students = await Student.find(filter)
      .sort({ createdAt: -1 })
      .skip((page - 1) * limit)
      .limit(Number(limit));

    const total = await Student.countDocuments(filter);
    res.json({ data: students, total, page: Number(page), pages:
Math.ceil(total / limit) });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}

export async function getStudentById(req, res) {
  try {
    const student = await Student.findById(req.params.id);
    if (!student) return res.status(404).json({ error: "Student not found"
});
    res.json(student);
```

```
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
}

export async function updateStudent(req, res) {
  try {
    const student = await Student.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true, runValidators: true }
    );
    if (!student) return res.status(404).json({ error: "Student not found"
});
    res.json(student);
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
}

export async function deleteStudent(req, res) {
  try {
    const student = await Student.findByIdAndDelete(req.params.id);
    if (!student) return res.status(404).json({ error: "Student not found"
});
    res.json({ message: "Student deleted" });
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
}
```

10) Routes (src/routes/students.js)

```
import { Router } from "express";
import {
  createStudent,
  getStudents,
  getStudentById,
  updateStudent,
  deleteStudent,
```

```
} from "../controllers/studentsController.js";

const router = Router();

router.post("/", createStudent);        // Create
router.get("/", getStudents);           // Read (list with
search/pagination)
router.get("/:id", getStudentById);     // Read (single)
router.put("/:id", updateStudent);      // Update (full)
router.patch("/:id", updateStudent);    // Update (partial)
router.delete("/:id", deleteStudent);   // Delete

export default router;
```

11) Server (src/server.js)

```
import express from "express";
import cors from "cors";
import dotenv from "dotenv";
import { connectDB } from "./db.js";
import studentRoutes from "./routes/students.js";

dotenv.config();
const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// Routes
app.use("/api/students", studentRoutes);

// Health check
app.get("/", (_req, res) => res.send("Student CRUD API is running"));

// Start
const PORT = process.env.PORT || 3000;
const URI = process.env.MONGODB_URI;

connectDB(URI).then(() => {
```

```
  app.listen(PORT, () => console.log(`🚀 Server listening on
http://localhost:${PORT}`));
});
```

12) Run the API

Start the server in development mode (auto-restart on changes):

npm run dev

If everything is fine, you should see a message like:

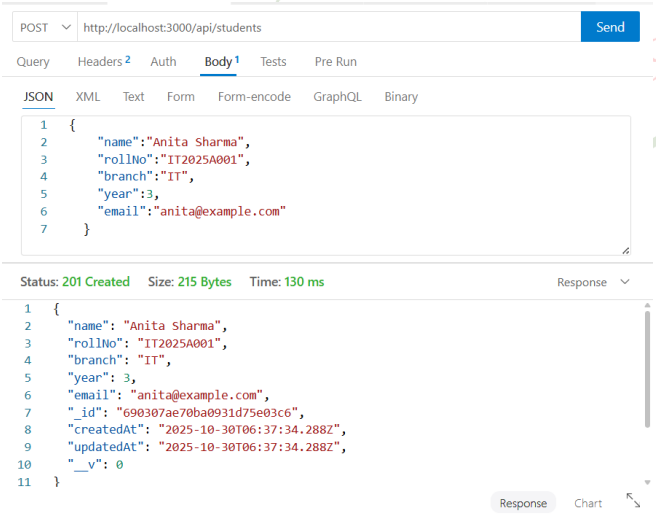
✅ Connected to MongoDB
🚀 Server listening on http://localhost:3000

13) Test the Endpoints

Use Postman/Thunder Client or these curl commands:

• Create a student:

```
POST http://localhost:3000/api/students \
"Content-Type: application/json" \
  '{
    "name":"Anita Sharma",
    "rollNo":"IT2025A001",
    "branch":"IT",
    "year":3,
    "email":"anita@example.com"}'
```



• List students (first page, 10 per page):

http://localhost:3000/api/students?page=1&limit=10

JSON XML Text Form Form-encode GraphQL Binary

```
{
    "name":"Anita Sharma",
    "rollNo":"IT2025A001",
    "branch":"IT",
    "year":3,
    "email":"anita@example.com"
}
```

Status: 201 Created Size: 215 Bytes Time: 130 ms

```
{
  "name": "Anita Sharma",
  "rollNo": "IT2025A001",
  "branch": "IT",
  "year": 3,
  "email": "anita@example.com",
  "_id": "690307ae70ba0931d75e03c6",
  "createdAt": "2025-10-30T06:37:34.288Z",
  "updatedAt": "2025-10-30T06:37:34.288Z",
  "__v": 0
}
```

• Search students (query by name/rollNo/branch):

http://localhost:3000/api/students?q=IT

GET http://localhost:3000/api/students?q=IT

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

```
{ "year": 4 }
```

Status: 200 OK Size: 255 Bytes Time: 11 ms

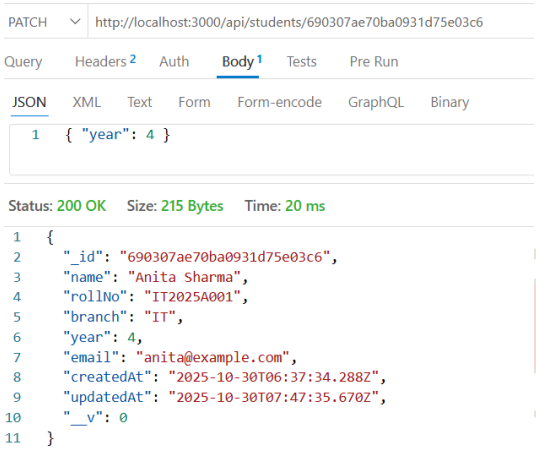
```
{
  "data": [
    {
      "_id": "690307ae70ba0931d75e03c6",
      "name": "Anita Sharma",
      "rollNo": "IT2025A001",
      "branch": "IT",
      "year": 3,
      "email": "anita@example.com",
      "createdAt": "2025-10-30T06:37:34.288Z",
      "updatedAt": "2025-10-30T06:37:34.288Z",
```

• Get one student by ID:

http://localhost:3000/api/students/<id>

• Update a student:

PATCH http://localhost:3000/api/students/<id> \
"Content-Type: application/json" \
- '{ "year": 4 }'



• Delete a student:

DELETE http://localhost:3000/api/students/<id>