Name : Suraj Gitte
Subject : CNT
Div : CSE(AI) - 35
Assingement : Design and develop a responsive website to prepare one semester result of VIT students using REACT, Springboot and MySQL/ MongoDB/Oracle. Take any four subjects with MSE Marks (30%) ESE Marks (70%)

---

1. System design (high level)

- Frontend: React + Tailwind CSS (responsive UI).
- Backend: Spring Boot (Java) exposing REST endpoints.
- Database: MySQL (relational schema shown). Optionally MongoDB (document model) alternative explained.
- Data model: Student, Subject, Mark (MSE, ESE).
- Authentication: not included in this minimal prototype (can be added later)

2. Frontend Code

a) Addmarks.jsx

```
import { useState } from "react";

export default function AddMarks() {
  const [form, setForm] = useState({
    rollNo: "",
    subjectCode: "",
    mseMarks: "",
    eseMarks: "",
  });
  const [status, setStatus] = useState(null);
```

```
  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };
```

```
  const handleSubmit = (e) => {
    e.preventDefault();
    // basic validation
    const rollNo = String(form.rollNo).trim();
    const subjectCode = String(form.subjectCode).trim();
    const mse = Number(form.mseMarks);
    const ese = Number(form.eseMarks);
```

```
    if (!rollNo || !subjectCode) {
      setStatus({
        type: "error",
        message: "Roll No and Subject Code are required.",
      });
      return;
    }
    if (Number.isNaN(mse) || Number.isNaN(ese)) {
      setStatus({ type: "error", message: "Marks must be valid numbers." });
      return;
    }
    if (mse < 0 || mse > 30 || ese < 0 || ese > 70) {
      setStatus({ type: "error", message: "Marks out of allowed range." });
      return;
    }
```

```
    const payload = {
      rollNo,
      subjectCode,
      mseMarks: mse,
      eseMarks: ese,
    };
```

```
    setStatus({ type: "pending", message: "Sending..." });
```

```
    console.log(payload);
```

```
    fetch("http://localhost:8080/results/add-marks", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(payload),
    })
      .then((res) => {
        if (!res.ok) throw new Error("Failed to add marks");
        return res.json();
      })
      .then(() =>
        setStatus({ type: "success", message: "Marks added successfully." })
      )
      .catch((err) =>
        setStatus({ type: "error", message: err.message || "Request failed" })
      );
  };
```

```
  return (
    <main className="text-black min-h-screen bg-gradient-to-br from-gray-50 to-
white p-6">
      <div className="max-w-2xl mx-auto">
        <div className="bg-white/80 backdrop-blur-sm shadow-xl rounded-2xl p-
8">
          <h1 className="text-3xl sm:text-4xl font-extrabold mb-6 text-gray-
900">
            Add Marks
```

```
          </h1>
```

```
          <form onSubmit={handleSubmit} className="space-y-6">
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                Roll No
              </label>
              <input
                className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
                name="rollNo"
                placeholder="Enter Roll No (e.g. 21BCE1234)"
                onChange={handleChange}
                required
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                Subject Code
              </label>
              <input
                className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
                name="subjectCode"
                placeholder="Enter Subject Code (e.g. CS501)"
                onChange={handleChange}
                required
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                MSE Marks (0-30)
              </label>
              <input
                className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
                name="mseMarks"
                type="number"
                min="0"
                max="30"
                placeholder="Enter MSE marks"
                onChange={handleChange}
                required
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
```

```
              ESE Marks (0-70)
            </label>
            <input
              className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
              name="eseMarks"
              type="number"
              min="0"
              max="70"
              placeholder="Enter ESE marks"
              onChange={handleChange}
              required
            />
          </div>
```

```
          {status && (
            <div
              className={`px-4 py-2 rounded-md text-sm ${
                status.type === "error"
                  ? "bg-red-100 text-red-800"
                  : status.type === "success"
                  ? "bg-green-100 text-green-800"
                  : "bg-yellow-100 text-yellow-800"
              }`}
            >
              {status.message}
            </div>
          )}
```

```
          <button
            type="submit"
            className="w-full px-6 py-3 bg-gradient-to-r from-blue-500 to-
blue-600 text-white font-medium rounded-xl hover:from-blue-600 hover:to-blue-
700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-400
transition-all transform hover:-translate-y-0.5 shadow-sm"
          >
            Add Marks
          </button>
        </form>
      </div>
    </div>
  </main>
 );
}
```

B) Add Students.jsx

```
import { useState } from "react";

export default function AddStudent() {
```

```
const [form, setForm] = useState({
  rollNo: "",
  firstName: "",
  lastName: "",
  branch: "",
  semester: "",
});
```

```
const [loading, setLoading] = useState(false);
const [message, setMessage] = useState(null);
```

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setForm((prev) => ({ ...prev, [name]: value }));
};
```

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setLoading(true);
  setMessage(null);
```

```
  try {
    const response = await fetch("http://localhost:8080/students/add", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(form),
    });
```

```
    if (!response.ok) {
      throw new Error("Failed to add student");
    }
```

```
    await response.json();
    setMessage({ type: "success", text: "Student added successfully!" });
```

```
    // reset form
    setForm({
      rollNo: "",
      firstName: "",
      lastName: "",
      branch: "",
      semester: "",
    });
  } catch (err) {
    setMessage({ type: "error", text: err.message });
  } finally {
    setLoading(false);
  }
};
```

```
return (
  <main className="min-h-screen bg-gradient-to-br from-gray-50 to-white p-6">
    <div className="max-w-2xl mx-auto">
```

```
        <div className="bg-white/80 backdrop-blur-sm shadow-xl rounded-2xl p-
8">
          <h1 className="text-3xl sm:text-4xl font-extrabold mb-6 text-gray-
900">
            Add Student
          </h1>
```

```
          {message && (
            <div
              className={`mb-4 p-3 rounded-lg text-sm ${
                message.type === "success"
                  ? "bg-green-100 text-green-800 border border-green-300"
                  : "bg-red-100 text-red-800 border border-red-300"
              }`}
            >
              {message.text}
            </div>
          )}
```

```
          <form onSubmit={handleSubmit} className="space-y-6 text-black">
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                Roll Number
              </label>
              <input
                type="text"
                name="rollNo"
                value={form.rollNo}
                onChange={handleChange}
                placeholder="Enter PRN Number"
                required
                className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                First Name
              </label>
              <input
                type="text"
                name="firstName"
                value={form.firstName}
                onChange={handleChange}
                placeholder="Enter First Name"
                required
                className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                Last Name
              </label>
              <input
                type="text"
                name="lastName"
                value={form.lastName}
                onChange={handleChange}
                placeholder="Enter Last Name"
                required
                className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
              />
            </div>
```

```
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                Branch
              </label>
              <select
                name="branch"
                value={form.branch}
                onChange={handleChange}
                required
                className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
              >
                <option value="">Select Branch</option>
                <option value="COMPUTER_SCIENCE">Computer Science</option>
                <option value="INFORMATION_TECHNOLOGY">
                  Information Technology
                </option>
                <option value="CSE_AI">CSE (AI)</option>
                <option value="ELECTRONICS">Electronics</option>
                <option value="MECHANICAL">Mechanical</option>
                <option value="CIVIL">Civil</option>
              </select>
            </div>
```

```
            <div>
              <label className="block text-sm font-medium text-gray-700 mb-2">
                Semester
              </label>
              <select
                name="semester"
                value={form.semester}
                onChange={handleChange}
                required
```

```
              className="w-full px-4 py-3 border border-gray-300 rounded-xl
focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-transparent
transition-all"
            >
              <option value="">Select Semester</option>
              <option value="FIRST">Semester 1</option>
              <option value="SECOND">Semester 2</option>
              <option value="THIRD">Semester 3</option>
              <option value="FOURTH">Semester 4</option>
              <option value="FIFTH">Semester 5</option>
              <option value="SIXTH">Semester 6</option>
              <option value="SEVENTH">Semester 7</option>
              <option value="EIGHTH">Semester 8</option>
            </select>
          </div>
```

```
          <button
            type="submit"
            disabled={loading}
            className="w-full px-6 py-3 bg-gradient-to-r from-blue-500 to-
indigo-600 text-white font-medium rounded-xl hover:from-blue-600 hover:to-
indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-
indigo-400 transition-all transform hover:-translate-y-0.5 shadow-sm
disabled:opacity-70 disabled:cursor-not-allowed"
          >
            {loading ? "Adding..." : "Add Student"}
          </button>
        </form>
      </div>
    </div>
  </main>
 );
}
```

ResultPage.jsx

```
import { useState } from "react";

export default function ResultPage() {
  const [studentId, setStudentId] = useState("");
  const [results, setResults] = useState([]);
```

```
  const fetchResults = () => {
    fetch(`http://localhost:8080/results/student/${studentId}/details`)
      .then((res) => res.json())
      .then((data) => setResults(data))
      .catch((err) => console.error(err));
  };
```

```
  console.log(results);
```

```
  return (
    <main className="min-h-screen bg-gradient-to-br from-gray-50 to-white p-6">
```

```
      <div className="max-w-6xl mx-auto">
        <div className="bg-white/80 backdrop-blur-sm shadow-xl rounded-2xl p-
8">
          <h1 className="text-3xl sm:text-4xl font-extrabold mb-6 text-gray-
900">
            Student Result
          </h1>
```

```
          <div className="flex flex-col sm:flex-row gap-4 mb-6">
            <input
              className=" text-black flex-1 px-4 py-3 border border-gray-300
rounded-xl focus:outline-none focus:ring-2 focus:ring-sky-500 focus:border-
transparent transition-all"
              placeholder="Enter Student ID"
              value={studentId}
              onChange={(e) => setStudentId(e.target.value)}
            />
            <button
              onClick={fetchResults}
              className="px-6 py-3 bg-gradient-to-r from-sky-500 to-indigo-600
text-white font-medium rounded-xl hover:from-sky-600 hover:to-indigo-700
focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-400
transition-all transform hover:-translate-y-0.5 shadow-sm"
            >
              Get Result
            </button>
          </div>
```

```
          {results.length > 0 && (
            <div className="overflow-x-auto">
              <table className="w-full border-collapse text-black bg-white
rounded-xl shadow-lg overflow-hidden">
                <thead>
                  <tr className="bg-gradient-to-r from-sky-500 to-indigo-600
text-white">
                    <th className="p-4 text-left font-semibold">Subject</th>
                    <th className="p-4 text-left font-semibold">MSE</th>
                    <th className="p-4 text-left font-semibold">ESE</th>
                    <th className="p-4 text-left font-semibold">Total</th>
                    <th className="p-4 text-left font-semibold">Grade</th>
                  </tr>
                </thead>
                <tbody>
                  {results.map((r, index) => (
                    <tr
                      key={r.result_id}
                      className={`${
                        index % 2 === 0 ? "bg-gray-50" : "bg-white"
                      } hover:bg-gray-100 transition-colors`}
                    >
                      <td className="p-4 border-b border-gray-200 font-medium">
                        {r.subjectName}
                      </td>
```

```
                    <td className="p-4 border-b border-gray-200">
                      {r.mseMarks}
                    </td>
                    <td className="p-4 border-b border-gray-200">
                      {r.eseMarks}
                    </td>
                    <td className="p-4 border-b border-gray-200 font-
semibold">
                      {r.totalMarks}
                    </td>
                    <td className="p-4 border-b border-gray-200">
                      <span
                        className={`px-3 py-1 rounded-full text-sm font-
medium ${
                          r.grade === "A"
                            ? "bg-green-100 text-green-800"
                            : r.grade === "B"
                            ? "bg-blue-100 text-blue-800"
                            : r.grade === "C"
                            ? "bg-yellow-100 text-yellow-800"
                            : "bg-red-100 text-red-800"
                        }`}
                      >
                        {r.grade}
                      </span>
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        )}
```

```
        {results.length === 0 && studentId && (
          <div className="text-center py-8 text-gray-500">
            <p className="text-lg">No results found for this student</p>
          </div>
        )}
      </div>
    </div>
  </main>
  );
}
```

StudentList.jsx

```
import { useEffect, useState } from "react";

export default function StudentList() {
  const [students, setStudents] = useState([]);
```

```
  useEffect(() => {
```

```
    fetch("http://localhost:8080/students")
      .then((res) => res.json())
      .then((data) => setStudents(data))
      .catch((err) => console.error(err));
  }, [students]);
```

```
  console.log(students);
```

```
  return (
    <main className="min-h-screen bg-gradient-to-br from-gray-50 to-white p-6">
      <div className="max-w-6xl mx-auto">
        <div className="bg-white/80 backdrop-blur-sm shadow-xl rounded-2xl p-
8">
          <h1 className="text-3xl sm:text-4xl font-extrabold mb-6 text-gray-
900">
            Students
          </h1>
```

```
          <div className="overflow-x-auto">
            <table className="w-full border-collapse bg-white rounded-xl
shadow-lg overflow-hidden">
              <thead>
                <tr className="bg-gradient-to-r from-sky-500 to-indigo-600
text-white">
                  <th className="p-4 text-left font-semibold">PRN Number</th>
                  <th className="p-4 text-left font-semibold">Name</th>
                  <th className="p-4 text-left font-semibold">Branch</th>
                  <th className="p-4 text-left font-semibold">Semester</th>
                </tr>
              </thead>
              <tbody className="text-gray-950">
                {students.map((s, index) => (
                  <tr
                    key={s.student_id}
                    className={`${
                      index % 2 === 0 ? "bg-gray-50" : "bg-white"
                    } hover:bg-gray-100 transition-colors `}
                  >
                    <td className="p-4 border-b border-gray-200 font-medium">
                      {s.rollNo}
                    </td>
                    <td className="p-4 border-b border-gray-200">
                      {s.firstName} {s.lastName}
                    </td>
                    <td className="p-4 border-b border-gray-
200">{s.branch}</td>
                    <td className="p-4 border-b border-gray-200">
                      {s.semester}
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
```

```
          </div>

          {students.length === 0 && (
            <div className="text-center py-8 text-gray-500">
              <p className="text-lg">No students found</p>
            </div>
          )}
        </div>
      </div>
    </main>
  );
}
```

---

3. Database Schema

a) Result.java

```
package com.vit.studentresultsystem.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import javax.security.auth.Subject;
import java.math.BigDecimal;
import java.time.LocalDateTime;

@Entity
@Table(name = "results",
        uniqueConstraints = @UniqueConstraint(columnNames = {"student_id",
"subject_id"}))
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Result {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    @Column(name = "result_id")
    private Integer resultId;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "student_id", nullable = false)
    private Student student;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "subject_id", nullable = false)
    private Subject subject;

    @Column(name = "mse_marks", nullable = false, precision = 5, scale = 2)
    private BigDecimal mseMarks;

    @Column(name = "ese_marks", nullable = false, precision = 5, scale = 2)
    private BigDecimal eseMarks;

    // Computed fields (as per database schema)
    @Column(name = "total_marks", insertable = false, updatable = false, precision = 5,
scale = 2)
    private BigDecimal totalMarks;

    @Column(name = "grade", insertable = false, updatable = false, length = 2)
    private String grade;

    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    // Helper methods for calculation (if needed in Java)
    public BigDecimal calculateTotalMarks() {
        return mseMarks.add(eseMarks);
    }

    public String calculateGrade() {
        BigDecimal total = calculateTotalMarks();
        if (total.compareTo(new BigDecimal("90")) >= 0) return "A+";
```

```
        if (total.compareTo(new BigDecimal("80")) >= 0) return "A";
        if (total.compareTo(new BigDecimal("70")) >= 0) return "B+";
        if (total.compareTo(new BigDecimal("60")) >= 0) return "B";
        if (total.compareTo(new BigDecimal("50")) >= 0) return "C+";
        if (total.compareTo(new BigDecimal("40")) >= 0) return "C";
        if (total.compareTo(new BigDecimal("35")) >= 0) return "D";
        return "F";
    }
}
```

B) Student. java

```
package com.vit.studentresultsystem.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "student")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String prn;

    private String first_name;
    private String last_name;
    private String branch;
    private String semester;

    // Getters & Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getRoll_no() { return prn; }
    public void setRoll_no(String roll_no) { this.prn = roll_no; }

    public String getFirst_name() { return first_name; }
    public void setFirst_name(String first_name) { this.first_name = first_name; }
```

```
    public String getLast_name() { return last_name; }
    public void setLast_name(String last_name) { this.last_name = last_name; }

    public String getBranch() { return branch; }
    public void setBranch(String branch) { this.branch = branch; }

    public String getSemester() { return semester; }
    public void setSemester(String semester) { this.semester = semester; }
}
```

Subject. java

```
package com.vit.studentresultsystem.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name = "subjects")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Subject {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "subject_id")
    private Integer subjectId;

    @Column(name = "subject_code", unique = true, nullable = false, length = 10)
    private String subjectCode;

    @Column(name = "subject_name", nullable = false, length = 100)
```

```
    private String subjectName;

    @Enumerated(EnumType.STRING)
    @Column(name = "branch", nullable = false)
    private Student.branch branch;

    @Enumerated(EnumType.STRING)
    @Column(name = "semester", nullable = false)
    private Student.semester semester;

    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;

    @UpdateTimestamp
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    @OneToMany(mappedBy = "subject", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private List<Result> results = new ArrayList<>();
}
```

---

4. Backend

A) StudentController

```
package com.vit.studentresultsystem.controller;

import com.vit.studentresultsystem.entity.Student;
import com.vit.studentresultsystem.repository.StudentRepository;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
```

```java
@RequestMapping("/api/students")
@CrossOrigin(origins = "http://localhost:5173")
public class StudentController {

    private final StudentRepository repo;

    public StudentController(StudentRepository repo) {
        this.repo = repo;
    }

    // Get all students
    @GetMapping
    public List<Student> getAllStudents() {
        return repo.findAll();
    }

    // Add new student
    @PostMapping
    public ResponseEntity<?> addStudent(@RequestBody Student student) {
        if (repo.existsByPrn(student.getRoll_no())) {
            return ResponseEntity.badRequest().body("Roll Number already exists!");
        }
        return ResponseEntity.ok(repo.save(student));
    }

    // Get student by id
    @GetMapping("/{id}")
    public ResponseEntity<Student> getStudent(@PathVariable Long id) {
        return repo.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    // Update student
    @PutMapping("/{id}")
    public ResponseEntity<Student> updateStudent(@PathVariable Long id,
@RequestBody Student details) {
        return repo.findById(id).map(student -> {
            student.setRoll_no(details.getRoll_no());
            student.setFirst_name(details.getFirst_name());
            student.setLast_name(details.getLast_name());
```

```
            student.setBranch(details.getBranch());
            student.setSemester(details.getSemester());
            return ResponseEntity.ok(repo.save(student));
        }).orElse(ResponseEntity.notFound().build());
    }

    // Delete student
    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteStudent(@PathVariable Long id) {
        if (!repo.existsById(id)) {
            return ResponseEntity.notFound().build();
        }
        repo.deleteById(id);
        return ResponseEntity.ok().build();
    }
}
```

B) StudentResultApplication.java

```
package com.vit.studentresultsystem;

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

@SpringBootApplication
public class StudentResultSystemApplication {

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

C) ResultConTroller.java

```
package com.vit.assingement02_cnt_backned.controller;

import com.vit.assingement02_cnt_backned.dto.ResultCreateRequest;
```

```java
import com.vit.assingement02_cnt_backned.dto.ResultResponse;
import com.vit.assingement02_cnt_backned.dto.StudentResultResponse;
import com.vit.assingement02_cnt_backned.entity.Result;
import com.vit.assingement02_cnt_backned.entity.Student;
import com.vit.assingement02_cnt_backned.service.ResultService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/results")
@CrossOrigin(origins = "http://localhost:5173")
public class ResultController {

    @Autowired
    private ResultService resultService;

    @GetMapping
    public ResponseEntity<List<Result>> getAllResults() {
        try {
            List<Result> results = resultService.getAllResults();
            return ResponseEntity.ok(results);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<Result> getResultById(@PathVariable Integer id) {
        try {
            return resultService.getResultById(id)
                    .map(result -> ResponseEntity.ok(result))
                    .orElse(ResponseEntity.notFound().build());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }
```

```
    @GetMapping("/student/{studentId}")
    public ResponseEntity<List<Result>> getResultsByStudentId(@PathVariable String studentId) {
        try {
            List<Result> results = resultService.getResultsByStudentId(studentId);
            return ResponseEntity.ok(results);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    // NEW ENDPOINT: Get comprehensive student results for frontend
    @GetMapping("/student/{rollNo}/details")
    public ResponseEntity<List<StudentResultResponse>> getStudentResultsForFrontend(@PathVariable String rollNo) {
        try {
            List<StudentResultResponse> results = resultService.getStudentResultsForFrontend(rollNo);
            return ResponseEntity.ok(results);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/subject/{subjectId}")
    public ResponseEntity<List<Result>> getResultsBySubjectId(@PathVariable String subjectId) {
        try {
            List<Result> results = resultService.getResultsBySubjectId(subjectId);
            return ResponseEntity.ok(results);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/student/roll/{rollNo}")
    public ResponseEntity<List<Result>> getResultsByStudentRollNo(@PathVariable String rollNo) {
        try {
            List<Result> results = resultService.getResultsByStudentRollNo(rollNo);
```

```
            return ResponseEntity.ok(results);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/subject/code/{subjectCode}")
    public ResponseEntity<List<Result>> getResultsBySubjectCode(@PathVariable
String subjectCode) {
        try {
            List<Result> results = resultService.getResultsBySubjectCode(subjectCode);
            return ResponseEntity.ok(results);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/branch/{branch}/semester/{semester}")
    public ResponseEntity<List<Result>> getResultsByBranchAndSemester(
            @PathVariable Student.Branch branch,
            @PathVariable Student.Semester semester) {
        try {
            List<Result> results = resultService.getResultsByBranchAndSemester(branch,
semester);
            return ResponseEntity.ok(results);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/student/{studentId}/subject/{subjectId}")
    public ResponseEntity<Result> getResultByStudentAndSubject(
            @PathVariable String studentId,
            @PathVariable String subjectId) {
        try {
            return resultService.getResultByStudentAndSubject(studentId, subjectId)
                    .map(result -> ResponseEntity.ok(result))
                    .orElse(ResponseEntity.notFound().build());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
```

```
    }

    // NEW ENDPOINT: Create result using roll number and subject code
    @PostMapping("/add-marks")
    public ResponseEntity<?> createResultByRollNoAndSubjectCode(@Valid
@RequestBody ResultCreateRequest request) {
        try {
            ResultResponse createdResult =
resultService.createResultByRollNoAndSubjectCodeAsDTO(request);
            return ResponseEntity.status(HttpStatus.CREATED).body(createdResult);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Error creating result: " + e.getMessage());
        }
    }

    // NEW ENDPOINT: Update result using roll number and subject code
    @PutMapping("/update-marks/{rollNo}/{subjectCode}")
    public ResponseEntity<?> updateResultByRollNoAndSubjectCode(
            @PathVariable String rollNo,
            @PathVariable String subjectCode,
            @Valid @RequestBody ResultCreateRequest request) {
        try {
            ResultResponse updatedResult =
resultService.updateResultByRollNoAndSubjectCodeAsDTO(rollNo, subjectCode,
request);
            return ResponseEntity.ok(updatedResult);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Error updating result: " + e.getMessage());
        }
    }

    // NEW ENDPOINT: Delete result using roll number and subject code
    @DeleteMapping("/delete-marks/{rollNo}/{subjectCode}")
    public ResponseEntity<?> deleteResultByRollNoAndSubjectCode(
            @PathVariable String rollNo,
```

```
            @PathVariable String subjectCode) {
        try {
            resultService.deleteResultByRollNoAndSubjectCode(rollNo, subjectCode);
            return ResponseEntity.ok().body("Result deleted successfully for student " +
rollNo +
                    " and subject " + subjectCode);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Error deleting result: " + e.getMessage());
        }
    }

    // EXISTING ENDPOINTS (keeping for backward compatibility)
    @PostMapping
    public ResponseEntity<?> createResult(@Valid @RequestBody Result result) {
        try {
            Result createdResult = resultService.createResult(result);
            return ResponseEntity.status(HttpStatus.CREATED).body(createdResult);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Error creating result: " + e.getMessage());
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> updateResult(@PathVariable Integer id, @Valid
@RequestBody Result resultDetails) {
        try {
            Result updatedResult = resultService.updateResult(id, resultDetails);
            return ResponseEntity.ok(updatedResult);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Error updating result: " + e.getMessage());
        }
    }
```

```
    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteResult(@PathVariable Integer id) {
        try {
            resultService.deleteResult(id);
            return ResponseEntity.ok().body("Result deleted successfully");
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Error deleting result: " + e.getMessage());
        }
    }
}
```

3. SubjectConTroller.java

```
package com.vit.assingement02_cnt_backned.controller;

import com.vit.assingement02_cnt_backned.dto.SubjectCreateRequest;
import com.vit.assingement02_cnt_backned.entity.Subject;
import com.vit.assingement02_cnt_backned.entity.Student;
import com.vit.assingement02_cnt_backned.service.SubjectService;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/subjects")
@CrossOrigin(origins = "http://localhost:5173")
public class SubjectController {

    @Autowired
    private SubjectService subjectService;
```

```
    @GetMapping
    public ResponseEntity<List<Subject>> getAllSubjects() {
        try {
            List<Subject> subjects = subjectService.getAllSubjects();
            return ResponseEntity.ok(subjects);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<Subject> getSubjectById(@PathVariable String id) {
        try {
            return subjectService.getSubjectById(id)
                .map(subject -> ResponseEntity.ok(subject))
                .orElse(ResponseEntity.notFound().build());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/code/{subjectCode}")
    public ResponseEntity<Subject> getSubjectByCode(@PathVariable String
subjectCode) {
        try {
            return subjectService.getSubjectByCode(subjectCode)
                .map(subject -> ResponseEntity.ok(subject))
                .orElse(ResponseEntity.notFound().build());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/branch/{branch}")
    public ResponseEntity<List<Subject>> getSubjectsByBranch(@PathVariable
Student.Branch branch) {
        try {
            List<Subject> subjects = subjectService.getSubjectsByBranch(branch);
            return ResponseEntity.ok(subjects);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
```

```
        }
    }

    @GetMapping("/semester/{semester}")
    public ResponseEntity<List<Subject>> getSubjectsBySemester(@PathVariable
Student.Semester semester) {
        try {
            List<Subject> subjects = subjectService.getSubjectsBySemester(semester);
            return ResponseEntity.ok(subjects);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/branch/{branch}/semester/{semester}")
    public ResponseEntity<List<Subject>> getSubjectsByBranchAndSemester(
            @PathVariable Student.Branch branch,
            @PathVariable Student.Semester semester) {
        try {
            List<Subject> subjects =
subjectService.getSubjectsByBranchAndSemester(branch, semester);
            return ResponseEntity.ok(subjects);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @GetMapping("/search")
    public ResponseEntity<List<Subject>> searchSubjectsByName(@RequestParam
String name) {
        try {
            List<Subject> subjects = subjectService.searchSubjectsByName(name);
            return ResponseEntity.ok(subjects);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }

    @PostMapping("/add")
    public ResponseEntity<?> createSubjectFromRequest(@Valid @RequestBody
SubjectCreateRequest request) {
```

```
        try {
            Subject createdSubject = subjectService.createSubjectFromRequest(request);
            return ResponseEntity.status(HttpStatus.CREATED).body(createdSubject);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Error creating subject: " + e.getMessage());
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> updateSubject(@PathVariable String id, @Valid
@RequestBody Subject subjectDetails) {
        try {
            Subject updatedSubject = subjectService.updateSubject(id, subjectDetails);
            return ResponseEntity.ok(updatedSubject);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Error updating subject: " + e.getMessage());
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteSubject(@PathVariable String id) {
        try {
            subjectService.deleteSubject(id);
            return ResponseEntity.ok().body("Subject deleted successfully");
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body("Error: " + e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Error deleting subject: " + e.getMessage());
        }
    }
}
```

E) ReSultService.java

```
package com.vit.assingement02_cnt_backned.service;

import com.vit.assingement02_cnt_backned.dto.ResultCreateRequest;
import com.vit.assingement02_cnt_backned.dto.ResultResponse;
import com.vit.assingement02_cnt_backned.dto.StudentResultResponse;
import com.vit.assingement02_cnt_backned.entity.Result;
import com.vit.assingement02_cnt_backned.entity.Student;
import com.vit.assingement02_cnt_backned.entity.Subject;
import com.vit.assingement02_cnt_backned.repository.ResultRepository;
import com.vit.assingement02_cnt_backned.repository.StudentRepository;
import com.vit.assingement02_cnt_backned.repository.SubjectRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class ResultService {

    @Autowired
    private ResultRepository resultRepository;

    @Autowired
    private StudentRepository studentRepository;

    @Autowired
    private SubjectRepository subjectRepository;

    // Helper method to convert Result entity to ResultResponse DTO
    private ResultResponse convertToResultResponse(Result result) {
        Subject subject = result.getSubject();
        return new ResultResponse(
```

```
            result.getResultId(),
            result.getStudentId(),
            subject != null ? subject.getSubjectId() : null,
            subject != null ? subject.getSubjectCode() : null,
            subject != null ? subject.getSubjectName() : null,
            result.getMseMarks(),
            result.getEseMarks(),
            result.getTotalMarks(),
            result.getGrade(),
            result.getCreatedAt(),
            result.getUpdatedAt()
        );
    }

    public List<Result> getAllResults() {
        return resultRepository.findAll();
    }

    public List<ResultResponse> getAllResultsAsDTO() {
        return resultRepository.findAll().stream()
                .map(this::convertToResultResponse)
                .collect(Collectors.toList());
    }

    public Optional<Result> getResultById(Integer id) {
        return resultRepository.findById(id);
    }

    public Optional<ResultResponse> getResultByIdAsDTO(Integer id) {
        return resultRepository.findById(id)
                .map(this::convertToResultResponse);
    }

    public List<Result> getResultsByStudentId(String studentId) {
        return resultRepository.findByStudentId(studentId);
    }

    // NEW METHOD: Get comprehensive student result data for frontend
    public List<StudentResultResponse> getStudentResultsForFrontend(String rollNo) {
        List<Result> results = resultRepository.findByStudentId(rollNo);
```

```
        return results.stream().map(result -> {
            Student student = result.getStudent();
            Subject subject = result.getSubject();

            return new StudentResultResponse(
                result.getResultId(),
                result.getStudentId(),
                student != null ? student.getRollNo() : result.getStudentId(),
                student != null ? student.getFirstName() : null,
                student != null ? student.getLastName() : null,
                student != null ? student.getBranch() : null,
                student != null ? student.getSemester() : null,
                subject != null ? subject.getSubjectId() : null,
                subject != null ? subject.getSubjectCode() : null,
                subject != null ? subject.getSubjectName() : null,
                result.getMseMarks(),
                result.getEseMarks(),
                result.getTotalMarks(),
                result.getGrade(),
                result.getCreatedAt(),
                result.getUpdatedAt()
            );
        }).collect(Collectors.toList());
    }

    public List<Result> getResultsBySubjectId(String subjectId) {
        return resultRepository.findBySubjectSubjectId(subjectId);
    }

    public List<Result> getResultsByStudentRollNo(String rollNo) {
        return resultRepository.findByStudentRollNo(rollNo);
    }

    public List<Result> getResultsBySubjectCode(String subjectCode) {
        return resultRepository.findBySubjectCode(subjectCode);
    }

    public List<Result> getResultsByBranchAndSemester(Student.Branch branch,
Student.Semester semester) {
        return resultRepository.findByBranchAndSemester(branch, semester);
    }
```

```
    public Optional<Result> getResultByStudentAndSubject(String studentId, String
subjectId) {
        return resultRepository.findByStudentIdAndSubjectSubjectId(studentId,
subjectId);
    }

    // NEW METHOD: Create result using roll number and subject code
    public Result createResultByRollNoAndSubjectCode(ResultCreateRequest request) {
        // Find student by roll number
        Student student = studentRepository.findByRollNo(request.getRollNo())
                .orElseThrow(() -> new RuntimeException("Student not found with roll
number: " + request.getRollNo()));

        // Find subject by subject code
        Subject subject = subjectRepository.findBySubjectCode(request.getSubjectCode())
                .orElseThrow(() -> new RuntimeException("Subject not found with code:
" + request.getSubjectCode()));

        // Check if result already exists for this student-subject combination
        if (resultRepository.findByStudentIdAndSubjectSubjectId(request.getRollNo(),
subject.getSubjectId()).isPresent()) {
            throw new RuntimeException("Result already exists for student " +
request.getRollNo() +
                    " and subject " + request.getSubjectCode());
        }

        // Create new result
        Result result = new Result();
        result.setStudentId(request.getRollNo());
        result.setSubject(subject);
        result.setMseMarks(request.getMseMarks());
        result.setEseMarks(request.getEseMarks());

        return resultRepository.save(result);
    }

    // NEW METHOD: Create result using roll number and subject code (returns DTO)
    public ResultResponse
createResultByRollNoAndSubjectCodeAsDTO(ResultCreateRequest request) {
        Result result = createResultByRollNoAndSubjectCode(request);
```

```
        return convertToResultResponse(result);
    }

    // UPDATED METHOD: Update result using roll number and subject code
    public Result updateResultByRollNoAndSubjectCode(String rollNo, String
subjectCode, ResultCreateRequest request) {
        // Find student by roll number
        Student student = studentRepository.findByRollNo(rollNo)
                .orElseThrow(() -> new RuntimeException("Student not found with roll
number: " + rollNo));

        // Find subject by subject code
        Subject subject = subjectRepository.findBySubjectCode(subjectCode)
                .orElseThrow(() -> new RuntimeException("Subject not found with code:
" + subjectCode));

        // Find existing result
        Result result = resultRepository.findByStudentIdAndSubjectSubjectId(rollNo,
subject.getSubjectId())
                .orElseThrow(() -> new RuntimeException("Result not found for student
" + rollNo +
                        " and subject " + subjectCode));

        // Update marks
        result.setMseMarks(request.getMseMarks());
        result.setEseMarks(request.getEseMarks());

        return resultRepository.save(result);
    }

    // NEW METHOD: Update result using roll number and subject code (returns DTO)
    public ResultResponse updateResultByRollNoAndSubjectCodeAsDTO(String rollNo,
String subjectCode, ResultCreateRequest request) {
        Result result = updateResultByRollNoAndSubjectCode(rollNo, subjectCode,
request);
        return convertToResultResponse(result);
    }

    // EXISTING METHODS (keeping for backward compatibility)
    public Result createResult(Result result) {
        // Verify student exists by roll number
```

```
        Student student = studentRepository.findByRollNo(result.getStudentId())
                .orElseThrow(() -> new RuntimeException("Student not found"));

        // Verify subject exists
        Subject subject = subjectRepository.findById(result.getSubject().getSubjectId())
                .orElseThrow(() -> new RuntimeException("Subject not found"));

        // Check if result already exists for this student-subject combination
        if (resultRepository.findByStudentIdAndSubjectSubjectId(result.getStudentId(),
result.getSubject().getSubjectId()).isPresent()) {
            throw new RuntimeException("Result already exists for this student and
subject combination");
        }

        result.setSubject(subject);
        return resultRepository.save(result);
    }

    public Result updateResult(Integer id, Result resultDetails) {
        return resultRepository.findById(id)
                .map(result -> {
                    result.setMseMarks(resultDetails.getMseMarks());
                    result.setEseMarks(resultDetails.getEseMarks());
                    return resultRepository.save(result);
                })
                .orElseThrow(() -> new RuntimeException("Result not found with id: " +
id));
    }

    public void deleteResult(Integer id) {
        if (!resultRepository.existsById(id)) {
            throw new RuntimeException("Result not found with id: " + id);
        }
        resultRepository.deleteById(id);
    }

    // NEW METHOD: Delete result by roll number and subject code
    public void deleteResultByRollNoAndSubjectCode(String rollNo, String subjectCode)
{
        // Find student by roll number
        Student student = studentRepository.findByRollNo(rollNo)
```

```
                .orElseThrow(() -> new RuntimeException("Student not found with roll
number: " + rollNo));

        // Find subject by subject code
        Subject subject = subjectRepository.findBySubjectCode(subjectCode)
                .orElseThrow(() -> new RuntimeException("Subject not found with code:
" + subjectCode));

        // Find and delete result
        Result result = resultRepository.findByStudentIdAndSubjectSubjectId(rollNo,
subject.getSubjectId())
                .orElseThrow(() -> new RuntimeException("Result not found for student
" + rollNo +
                        " and subject " + subjectCode));

        resultRepository.delete(result);
    }
}
```

F) StudentService.java

```
package com.vit.assingement02_cnt_backned.service;

import com.vit.assingement02_cnt_backned.dto.StudentCreateRequest;
import com.vit.assingement02_cnt_backned.entity.Student;
import com.vit.assingement02_cnt_backned.repository.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class StudentService {

    @Autowired
    private StudentRepository studentRepository;

    public List<Student> getAllStudents() {
```

```
        return studentRepository.findAll();
    }

    public Optional<Student> getStudentById(Integer id) {
        return studentRepository.findById(id);
    }

    public Optional<Student> getStudentByRollNo(String rollNo) {
        return studentRepository.findByRollNo(rollNo);
    }

    public List<Student> getStudentsByBranch(Student.Branch branch) {
        return studentRepository.findByBranch(branch);
    }

    public List<Student> getStudentsBySemester(Student.Semester semester) {
        return studentRepository.findBySemester(semester);
    }

    public List<Student> getStudentsByBranchAndSemester(Student.Branch branch,
Student.Semester semester) {
        return studentRepository.findByBranchAndSemester(branch, semester);
    }

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

    public Student createStudent(Student student) {
        System.out.println(student);
        if (studentRepository.existsByRollNo(student.getRollNo())) {
            throw new RuntimeException("Student with roll number " +
student.getRollNo() + " already exists");
        }
        return studentRepository.save(student);
    }

    public Student updateStudent(Integer id, Student studentDetails) {
        return studentRepository.findById(id)
            .map(student -> {
                if (!student.getRollNo().equals(studentDetails.getRollNo()) &&
```

```
                studentRepository.existsByRollNo(studentDetails.getRollNo())) {
                throw new RuntimeException("Student with roll number " +
studentDetails.getRollNo() + " already exists");
            }
            student.setRollNo(studentDetails.getRollNo());
            student.setFirstName(studentDetails.getFirstName());
            student.setLastName(studentDetails.getLastName());
            student.setBranch(studentDetails.getBranch());
            student.setSemester(studentDetails.getSemester());
            return studentRepository.save(student);
        })
        .orElseThrow(() -> new RuntimeException("Student not found with id: " +
id));
    }

    public Student createStudentFromRequest(StudentCreateRequest request) {
        Student student = new Student();
        student.setRollNo(request.getRollNo());
        student.setFirstName(request.getFirstName());
        student.setLastName(request.getLastName());
        student.setBranch(request.getBranch());
        student.setSemester(request.getSemester());
        return createStudent(student);
    }

    public void deleteStudent(Integer id) {
        if (!studentRepository.existsById(id)) {
            throw new RuntimeException("Student not found with id: " + id);
        }
        studentRepository.deleteById(id);
    }
}
```

G) SubjectService.java

```
package com.vit.assingement02_cnt_backned.service;

import com.vit.assingement02_cnt_backned.dto.SubjectCreateRequest;
import com.vit.assingement02_cnt_backned.entity.Subject;
```

```java
import com.vit.assingement02_cnt_backned.entity.Student;
import com.vit.assingement02_cnt_backned.repository.SubjectRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class SubjectService {

    @Autowired
    private SubjectRepository subjectRepository;

    public List<Subject> getAllSubjects() {
        return subjectRepository.findAll();
    }

    public Optional<Subject> getSubjectById(String id) {
        return subjectRepository.findById(id);
    }

    public Optional<Subject> getSubjectByCode(String subjectCode) {
        return subjectRepository.findBySubjectCode(subjectCode);
    }

    public List<Subject> getSubjectsByBranch(Student.Branch branch) {
        return subjectRepository.findByBranch(branch);
    }

    public List<Subject> getSubjectsBySemester(Student.Semester semester) {
        return subjectRepository.findBySemester(semester);
    }

    public List<Subject> getSubjectsByBranchAndSemester(Student.Branch branch,
Student.Semester semester) {
        return subjectRepository.findByBranchAndSemester(branch, semester);
    }

    public List<Subject> searchSubjectsByName(String subjectName) {
        return subjectRepository.findBySubjectNameContaining(subjectName);
```

```
    }

    public Subject createSubject(Subject subject) {
        if (subjectRepository.existsBySubjectCode(subject.getSubjectCode())) {
            throw new RuntimeException("Subject with code " + subject.getSubjectCode()
+ " already exists");
        }
        return subjectRepository.save(subject);
    }

    public Subject updateSubject(String id, Subject subjectDetails) {
        return subjectRepository.findById(id)
            .map(subject -> {
                if (!subject.getSubjectCode().equals(subjectDetails.getSubjectCode()) &&
                    subjectRepository.existsBySubjectCode(subjectDetails.getSubjectCode()))
{
                    throw new RuntimeException("Subject with code " +
subjectDetails.getSubjectCode() + " already exists");
                }
                subject.setSubjectCode(subjectDetails.getSubjectCode());
                subject.setSubjectName(subjectDetails.getSubjectName());
                subject.setBranch(subjectDetails.getBranch());
                subject.setSemester(subjectDetails.getSemester());
                return subjectRepository.save(subject);
            })
            .orElseThrow(() -> new RuntimeException("Subject not found with id: " +
id));
    }

    public Subject createSubjectFromRequest(SubjectCreateRequest request) {
        Subject subject = new Subject();
        subject.setSubjectId(request.getSubjectId());
        subject.setSubjectCode(request.getSubjectCode());
        subject.setSubjectName(request.getSubjectName());
        subject.setBranch(request.getBranch());
        subject.setSemester(request.getSemester());
        return createSubject(subject);
    }

    public void deleteSubject(String id) {
        if (!subjectRepository.existsById(id)) {
```

```
            throw new RuntimeException("Subject not found with id: " + id);
        }
        subjectRepository.deleteById(id);
    }
}
```