Техническое задание

Команда №2

Проект: Разработка модуля для управления данными студентов

1. Цель проекта:

Создать API, позволяющее добавлять, редактировать, получать, фильтровать и удалять информацию о студентах. **Основные задачи:**

1. Реализовать CRUD-операции для работы со студентами
2. Добавить фильтрацию по: (ИДЕНТИФИКАТОР, ФИО (частичный поиск), Группе Дате зачисления).
3. Разработать REST-эндпоинты
4. Подключить Swagger для документации
5. Настроить логирование операций

2. Функциональные требования

2.1. Основные функции

API должно будет поддерживать:

- **Добавление студента** (регистрация основных данных).
- **Просмотр списка студентов** (с фильтрацией).
- **Получение данных конкретного студента** (по ID).
- **Обновление информации** (изменение ФИО, группы и др.).
- **Удаление студента** (с подтверждением).

2.2. Фильтрация данных

Поддержка поиска по:

- **ID**
- **ФИО**
- **Группе**
- **Дате зачисления**

2.3. Эндпоинты

Метод	Путь	Действие	Параметры (опционально)
POST	/api/students	Добавление нового студента	fullName, group
GET	/api/ students	Получить всех студентов/отфильтрованных	id, fullName, group, enrollmentDate
GET	/api/ students/{id}	Получить студента по ID	—
PUT	/api/ students/{id}	Обновить данные студента	fullName, group
DELETE	/api/ students/{id}	Удалить студента	—

Начало работы

Student.java

```java
package com.simul_tech.netgenius.models;

import jakarta.persistence.*;
import lombok.Data;
import java.time.LocalDate;

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

    @Column(nullable = false)
    private String fullName;

    @Column(nullable = false)
    private String group;

    @Column(nullable = false)
    private LocalDate enrollmentDate;
}
```

StudentRepository.java

```java
package com.simul_tech.netgenius.repositories;

import com.simul_tech.netgenius.models.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.time.LocalDate;
import java.util.List;

@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {
    List<Student> findByFullNameContainingIgnoreCase(String partialName);
    List<Student> findByGroup(String groupName);
    List<Student> findByGroupContaining(String groupPart);
    List<Student> findByEnrollmentDateBetween(LocalDate startDate, LocalDate endDate);
    List<Student> findByFullNameAndGroup(String fullName, String group);
}
```

StudentService.java

```java
package com.simul_tech.netgenius.services;

import com.simul_tech.netgenius.models.Student;
import com.simul_tech.netgenius.repositories.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.time.LocalDate;
import java.util.List;
import java.util.Optional;

@Service
public class StudentService {
    private final StudentRepository studentRepository;

    @Autowired
    public StudentService(StudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }

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

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

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

    public Student updateStudent(Long id, Student studentDetails) {
        Student student = studentRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Student not found"));

        student.setFullName(studentDetails.getFullName());
        student.setGroup(studentDetails.getGroup());
        student.setEnrollmentDate(studentDetails.getEnrollmentDate());

        return studentRepository.save(student);
    }
```

```
    public void deleteStudent(Long id) {
        studentRepository.deleteById(id);
    }

    public List<Student> filterStudents(String fullName, String group, LocalDate startDate, LocalDate e
        if (fullName != null && group != null) {
            return studentRepository.findByFullNameAndGroup(fullName, group);
        } else if (fullName != null) {
            return studentRepository.findByFullNameContainingIgnoreCase(fullName);
        } else if (group != null) {
            return studentRepository.findByGroup(group);
        } else if (startDate != null && endDate != null) {
            return studentRepository.findByEnrollmentDateBetween(startDate, endDate);
        }
        return studentRepository.findAll();
    }
}
```

StudentController.java

```java
package com.simul_tech.netgenius.controllers;

import com.simul_tech.netgenius.models.Student;
import com.simul_tech.netgenius.services.StudentService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.time.LocalDate;
import java.util.List;

@RestController
@RequestMapping("/api/students")
@Tag(name = "Student Management")
public class StudentController {
    private final StudentService studentService;

    public StudentController(StudentService studentService) {
        this.studentService = studentService;
    }

    @PostMapping
    @Operation(summary = "Create new student")
    public ResponseEntity<Student> createStudent(@RequestBody Student student) {
        return new ResponseEntity<>(studentService.createStudent(student), HttpStatus.CREATED);
    }

    @GetMapping
    @Operation(summary = "Get all/filtered students")
    public ResponseEntity<List<Student>> getStudents(
            @RequestParam(required = false) String fullName,
            @RequestParam(required = false) String group,
            @RequestParam(required = false) LocalDate startDate,
            @RequestParam(required = false) LocalDate endDate) {
        return ResponseEntity.ok(studentService.filterStudents(fullName, group, startDate, endDate));
    }

    @GetMapping("/{id}")
    @Operation(summary = "Get student by ID")
    public ResponseEntity<Student> getStudent(@PathVariable Long id) {
        return studentService.getStudentById(id)
```

```
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    @PutMapping("/{id}")
    @Operation(summary = "Update student")
    public ResponseEntity<Student> updateStudent(@PathVariable Long id, @RequestBody Student student) {
        return ResponseEntity.ok(studentService.updateStudent(id, student));
    }

    @DeleteMapping("/{id}")
    @Operation(summary = "Delete student")
    public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
        return ResponseEntity.noContent().build();
    }
}
```