```
package com.cst438.controllers;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;

import com.cst438.domain.Assignment;
import com.cst438.domain.AssignmentListDTO;
import com.cst438.domain.AssignmentGrade;
import com.cst438.domain.AssignmentGradeRepository;
import com.cst438.domain.AssignmentRepository;
import com.cst438.domain.Course;
import com.cst438.domain.CourseDTOG;
import com.cst438.domain.CourseRepository;
import com.cst438.domain.Enrollment;
import com.cst438.domain.GradebookDTO;
import com.cst438.services.RegistrationService;

@RestController
@CrossOrigin(origins = {"http://localhost:3000","http://localhost:3001"})
public class GradeBookController {

	@Autowired
	AssignmentRepository assignmentRepository;

	@Autowired
	AssignmentGradeRepository assignmentGradeRepository;

	@Autowired
	CourseRepository courseRepository;

	@Autowired
	RegistrationService registrationService;

	// get assignments for an instructor that need grading
	@GetMapping("/gradebook")
	public AssignmentListDTO getAssignmentsNeedGrading( ) {

		String email = "dwisneski@csumb.edu";  // user name (should be instructor's email)

		List<Assignment> assignments = assignmentRepository.findNeedGradingByEmail(email);
		AssignmentListDTO result = new AssignmentListDTO();
		for (Assignment a: assignments) {
			result.assignments.add(new AssignmentListDTO.AssignmentDTO(a.getId(),
a.getCourse().getCourse_id(), a.getName(), a.getDueDate().toString() , a.getCourse().getTitle()));
```

```
        }
        return result;
    }

    @GetMapping("/gradebook/{id}")
    public GradebookDTO getGradebook(@PathVariable("id") Integer assignmentId  ) {

        String email = "dwisneski@csumb.edu";  // user name (should be instructor's email)
        Assignment assignment = checkAssignment(assignmentId, email);

        // get the enrollment for the course
        //  for each student, get the current grade for assignment,
        //   if the student does not have a current grade, create an empty grade
        GradebookDTO gradebook = new GradebookDTO();
        gradebook.assignmentId= assignmentId;
        gradebook.assignmentName = assignment.getName();
        for (Enrollment e : assignment.getCourse().getEnrollments()) {
            GradebookDTO.Grade grade = new GradebookDTO.Grade();
            grade.name = e.getStudentName();
            grade.email = e.getStudentEmail();
            // does student have a grade for this assignment
            AssignmentGrade ag =
assignmentGradeRepository.findByAssignmentIdAndStudentEmail(assignmentId,  grade.email);
            if (ag != null) {
                grade.grade = ag.getScore();
                grade.assignmentGradeId = ag.getId();
            } else {
                grade.grade = "";
                AssignmentGrade agNew = new AssignmentGrade(assignment, e);
                agNew = assignmentGradeRepository.save(agNew);
                grade.assignmentGradeId = agNew.getId();  // key value generated by database on
save.
            }
            gradebook.grades.add(grade);
        }
        return gradebook;
    }

    @PostMapping("/course/{course_id}/finalgrades")
    @Transactional
    public void calcFinalGrades(@PathVariable int course_id) {
        System.out.println("Gradebook - calcFinalGrades for course " + course_id);

        // check that this request is from the course instructor
        String email = "dwisneski@csumb.edu";  // user name (should be instructor's email)

        Course c = courseRepository.findById(course_id).orElse(null);
        if (!c.getInstructor().equals(email)) {
            throw new ResponseStatusException( HttpStatus.UNAUTHORIZED, "Not Authorized. " );
        }

        CourseDTOG cdto = new CourseDTOG();
        cdto.course_id = course_id;
        cdto.grades = new ArrayList<>();
        for (Enrollment e: c.getEnrollments()) {
            double total=0.0;
            int count = 0;
```

```
                for (AssignmentGrade ag : e.getAssignmentGrades()) {
                        count++;
                        total = total + Double.parseDouble(ag.getScore());
                }
                double average = total/count;
                CourseDTOG.GradeDTO gdto = new CourseDTOG.GradeDTO();
                gdto.grade=letterGrade(average);
                gdto.student_email=e.getStudentEmail();
                gdto.student_name=e.getStudentName();
                cdto.grades.add(gdto);
                System.out.println("Course="+course_id+" Student="+e.getStudentEmail()+"
grade="+gdto.grade);
        }

        registrationService.sendFinalGrades(course_id, cdto);
    }

    private String letterGrade(double grade) {
        if (grade >= 90) return "A";
        if (grade >= 80) return "B";
        if (grade >= 70) return "C";
        if (grade >= 60) return "D";
        return "F";
    }

    @PutMapping("/gradebook/{id}")
    @Transactional
    public void updateGradebook (@RequestBody GradebookDTO gradebook, @PathVariable("id") Integer
assignmentId ) {

        String email = "dwisneski@csumb.edu";  // user name (should be instructor's email)
        checkAssignment(assignmentId, email);  // check that user name matches instructor email of the
course.

        // for each grade in gradebook, update the assignment grade in database
        System.out.printf("%d %s %d\n",  gradebook.assignmentId, gradebook.assignmentName,
gradebook.grades.size());

        for (GradebookDTO.Grade g : gradebook.grades) {
                System.out.printf("%s\n", g.toString());
                AssignmentGrade ag =
assignmentGradeRepository.findById(g.assignmentGradeId).orElse(null);
                if (ag == null) {
                        throw new ResponseStatusException( HttpStatus.BAD_REQUEST, "Invalid grade
primary key. "+g.assignmentGradeId);
                }
                ag.setScore(g.grade);
                System.out.printf("%s\n", ag.toString());

                assignmentGradeRepository.save(ag);
        }

    }

    private Assignment checkAssignment(int assignmentId, String email) {
        // get assignment
        Assignment assignment = assignmentRepository.findById(assignmentId).orElse(null);
```

```
		if (assignment == null) {
			throw new ResponseStatusException( HttpStatus.BAD_REQUEST, "Assignment not found.
"+assignmentId );
		}
		// check that user is the course instructor
		if (!assignment.getCourse().getInstructor().equals(email)) {
			throw new ResponseStatusException( HttpStatus.UNAUTHORIZED, "Not Authorized. " );
		}

		return assignment;
	}

}
```