MACHINE TEST - L1

==========

School Management System

Objective:
Build a backend API using .NET Core Web API to manage students, teachers, courses, classes, attendance, and grading with role-based access and relational database design.

Functional Requirements:
1. Entities
i. User:
- Id, Name, Email, Password (hashed), Role (Admin / Teacher / Student), CreatedDate, UpdatedDate

ii. Department:
- Id, Name, Description, HeadOfDepartmentId (Teacher), CreatedDate, UpdatedDate

iii. Course:
- Id, Name, Code, Description, DepartmentId, Credits, CreatedDate, UpdatedDate
- Each course belongs to a department
- Each course can have multiple classes (batches)

iv. Class (Batch):
- Id, Name, CourseId, TeacherId, Semester, StartDate, EndDate, IsActive, CreatedDate, UpdatedDate
- A teacher teaches a class for a course
- A class contains multiple students

v. StudentClass (Mapping Table):
- Id, StudentId, ClassId, EnrollmentDate
- Represents enrollment of a student in a class

vi. Attendance:
- Id, ClassId, StudentId, Date, Status (Present/Absent/Late), MarkedByTeacherId, CreatedDate

vii. Assignment:
- Id, ClassId, Title, Description, DueDate, CreatedDate, CreatedByTeacherId

viii. Submission:
- Id, AssignmentId, StudentId, SubmittedDate, FileUrl, Grade, GradedByTeacherId, Remarks

ix. Notification (Bonus):
- Id, Title, Message, RecipientRole, RecipientId (optional), CreatedDate, IsRead

2. Authentication & Authorization
- JWT-based authentication
- Roles:
  - Admin: Manage users, departments, and courses
  - Teacher: Manage classes, attendance, assignments, and grades
  - Student: View courses, submit assignments, view grades
- Endpoints:
  - POST /api/auth/register → Register new user (admin/student/teacher)
  - POST /api/auth/login → Login and receive JWT
  - POST /api/auth/refresh-token → Refresh JWT token

3. Admin APIs
- CRUD for:
  - Departments → /api/admin/departments
  - Courses → /api/admin/courses
  - Users (optional) → /api/admin/users
- Validation rules:
  - Department names must be unique.
  - Course codes must be unique per department.
  - Assign only teachers as Head of Department.

4. Teacher APIs
- Manage Classes /api/teacher/classes
  - Create, update, or deactivate classes
  - Assign students to a class
- Attendance Management:
  - POST /api/teacher/attendance → Mark attendance
  - GET /api/teacher/attendance/{classId} → View attendance history
- Assignment Management:
  - POST /api/teacher/assignments → Create new assignment
  - GET /api/teacher/assignments/{classId}
  - POST /api/teacher/assignments/{id}/grade → Grade student submissions
- Notifications:
  - POST /api/teacher/notifications → Send notification to class/students

5. Student APIs
- View enrolled classes /api/student/classes
- View attendance /api/student/attendance
- View and submit assignments /api/student/assignments/{id}/submit
- View grades /api/student/grades
- Receive notifications /api/student/notifications

6. Data Validation Rules
- Email format & password strength validation
- Prevent duplicate enrollments of the same student in the same class
- Assignment due date cannot be in the past
- Only the assigned teacher can mark attendance or grade submissions
- Admin-only endpoints cannot be accessed by non-admin roles

7. Bonus Functionalities
- Async/await for DB operations
- In-memory caching for course list and department list
- Pagination and filtering for:
  - Classes, Students, and Assignments
- Logging (Serilog or NLog)
- Soft delete for users and courses (IsActive = false)
- Optional:
  - Email notification when assignment is graded or new class is created
  - File upload API for assignment submissions (using IFormFile)

8. Tech Stack Recommendations
- Backend: .NET Core Web API
- Database: SQL Server
- ORM: Entity Framework Core
- Authentication: JWT
- Optional Tools: AutoMapper, FluentValidation, Serilog

9. Submission Requirements
- Submit GitHub repository link
- Include README.md with:
  - Setup instructions
  - DB migration commands
  - Sample API requests (Swagger recommended)
  - Short demo video showing API flow (Admin → Teacher → Student usage)

Evaluation Criteria:
- API design, structure, and coding best practices
- Role-based authentication and authorization
- Complex entity relationships and proper use of EF Core navigation properties
- Validation and error handling

- Async programming and caching
- Bonus: Logging and notification system

Submission:
Please submit your completed solution as a GitHub repository link. Additionally, include a README file with setup instructions and send to tech.codeandcraft@gmail.com.