Software Architecture Exercise
School Management System

Purpose

Strengthen architectural thinking through a realistic, bounded system: a School Management system focusing on:

- Student enrollment by class
- Subjects and class offerings
- Grades per student

Trainees will model the domain, choose a suitable architecture style, and produce clear, actionable diagrams and documentation that would enable a small team to implement a first iteration.

Scope & Core Use Cases

In Scope (MVP)

1. Manage Subjects: Create/list subjects (e.g., Math, Literature). - MV
2. Manage Classes: Create/list classes which include subject and term (e.g., 1st grade · Term 1 · Teacher A). - FR - LT
3. Student Enrollment: Enroll/unenroll students into classes; capacity rules apply. - JR - MB
4. Record Grades: Enter and update grades per student per class; view grade history/audit. - AM- VC

5. View Progress: For a student, list classes and grades for a given term. - AG

Non-Functional Requirements (NFRs)

- Scalability: Scale reads independently from writes for grade reports.
- Security: Role-based access (Admin, Teacher, Student).
- Auditability: Changes to grades must be auditable (who/when/what changed).
- Data Integrity: Prevent enrollment beyond class capacity (20 students per class); grades are bounded (0–100).
- Observability: Basic tracing/metrics/logging with correlation IDs.

Constraints & Assumptions

- Tech-neutral at the architecture level; you may propose a recommended reference stack.
- Start with a modular monolith or simple service decomposition (explain trade-offs).
- Assume single region deployment; future multi-region is a stretch goal.
- Transactional consistency needed for enrollment and grade writes.
- Assume a relational database for core entities; a read model or cache is acceptable for reports.

Deliverables

1. Architecture Overview

- Component Diagram (the whole system's)
- Sequence Diagram (the whole system's and per module)
- Activity Diagram (the whole system's)
- Class Diagram (the whole system's and per module)

2. Data Model
    - ERD (tables + key constraints) and domain model (class diagram).
    - Data life cycle for grades (write → revise → read).
3. API Definition
    - REST (OpenAPI) or GraphQL SDL for core use cases.
4. Sequence Diagrams
    - Enroll Student in Class (capacity + pre-req validation).
    - Record/Update Grade (audit trail).
    - View Student Progress (read model/cache path, if applicable).
5. Operational View
    - Deployment diagram (environments), logging/metrics, backup/restore, secrets management.
    - Access control model (roles → permissions).
6. Testing Strategy
    - Unit, contract/API, integration, and performance testing plan aligned with NFRs.

Architecture Guidance (Expectations)

- Style: Split into services: Enrollment, Class Catalog, Grading, Student Registry, Identity/Access.
- Data: Central relational DB with clearly defined ownership per module. Consider a read-optimized projection for student progress.
- Security: JWT/OAuth2/OIDC with role checks at API gateway or app layer; field-level validation for grades.
- Observability: Structured logs, key metrics (RPS, latency p95, error rate).
- Resilience: Timeouts/retries for any external calls, idempotency for grade updates.

Suggested Domain & APIs

Entities (minimum)

- Student(id, full_name, email, status)
- Subject(id, code, name)
- Class(id, subject_id, term, teacher_id, schedule, capacity, status)
- Enrollment(id, student_id, class_id, enrolled_at, status)
- Grade(id, enrollment_id, value, graded_at, graded_by, reason, version)
- Teacher(id, full_name, email)

Suggested REST Endpoints

```
POST   /students
GET    /students/{id}
POST   /subjects
POST   /classes
GET    /classes?subject={code}&term={term}
POST   /enrollments        // body: student_id, class_id
DELETE /enrollments/{id}   // unenroll
POST   /grades             // body: enrollment_id, value, reason
PUT    /grades/{id}        // update with audit
GET    /students/{id}/progress?term={term}
```

Validation Rules

- Enrollment is allowed only if class.status=OPEN and seats available (20 per class).
- Grade.value ∈ [0, 100].
- Updates to Grade must create an audit trail (versioning).