```
@startuml
title School Management System — Component Diagram (Whole System)

skinparam componentStyle uml2
skinparam rectangle {
  BackgroundColor White
  BorderColor Black
}
skinparam databaseBorderColor Black

'=== External / Front Door ===
package "Front Door" {
  [UI / API Gateway] as APIGW
  interface IIdentityAndAccess
  APIGW -right-> IIdentityAndAccess : verifyToken()\nhasRole()
}

'=== Identity & Access ===
component "Identity & Access" as IDP
IIdentityAndAccess -down-> IDP

'=== Domain Modules (Services) ===
package "Domain Services" {
  ' Subjects
  interface ISubjectsService
  component "Subjects Management" as SUBJECTS
  SUBJECTS -up- ISubjectsService : provides

  ' Teachers
  interface ITeacherService
  component "Teacher Management" as TEACHERS
  TEACHERS -up- ITeacherService : provides

  ' Students
  interface IStudentService
  component "Student Registry" as STUDENTS
  STUDENTS -up- IStudentService : provides

  ' Classes / Catalog
  interface IClassService
  component "Class Catalog" as CLASSES
  CLASSES -up- IClassService : provides

  ' Enrollment
  interface IEnrollmentService
  component "Enrollment" as ENROLL
  ENROLL -up- IEnrollmentService : provides
  ' Grading
  interface IGradeService
  component "Grading" as GRADING
  GRADING -up- IGradeService : provides
}

'=== API Gateway consumes domain services ===
APIGW -down-> ISubjectsService : manage subjects
APIGW -down-> ITeacherService : manage teachers
APIGW -down-> IStudentService : create/get students
APIGW -down-> IClassService : create/list classes\n(filter by subject/term)
APIGW -down-> IEnrollmentService : enroll/unenroll
APIGW -down-> IGradeService : record/update grade

'=== Inter-service required ports (ports & adapters vibe) ===
(IStudentService) -left- ENROLL : required
(IClassService)   -up-   ENROLL : required
(IEnrollmentService) -left- GRADING : required

'=== Persistence / Read Model ===
database "Relational DB\n(owned tables per module)" as DB
rectangle "Read Model / Reports\n(optional projection for progress)" as READMODEL

' Repositories (implicit) — each module talks to its own schema/tables
ENROLL -down-> DB : Enrollment tables\n(atomic writes)
GRADING -down-> DB : Grade tables\n(version/audit)
CLASSES -down-> DB : Class tables
STUDENTS -down-> DB : Student tables
TEACHERS -down-> DB : Teacher tables
SUBJECTS -down-> DB : Subject tables

' Read model fed by ETL/events from core tables (optional)
ENROLL -right-> READMODEL : publish/enrich
GRADING -right-> READMODEL : publish/enrich
APIGW -down-> READMODEL : GET /students/{id}/progress?term=

'=== Notes for the key constraints/NFRs ===
note right of APIGW
Role-based access checks
(Admin / Teacher / Student)
end note

note bottom of ENROLL
Transactional consistency
for enrollment writes
(capacity, OPEN, seats>0)
end note
```

```
note bottom of GRADING
Transactional consistency for grade writes
0 ≤ value ≤ 100, versioned/audited updates
end note

@enduml

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


@startuml
title School Management — System Activity (Swimlanes + guards + TX)

skinparam shadowing false
skinparam ActivityBackgroundColor White
skinparam ActivityBorderColor Black
skinparam SwimlaneBorderColor Black
skinparam NoteBackgroundColor #ffffee

' ==================== LOGIN (Admin) ====================
|Admin|
start
:Tries to login;

|System|
:Check credentials;
if (Valid credentials?) then (Yes)
  :Return JWT;
else (No)
  :Return 401 Login Error;
  stop
endif

' ==================== ADMIN: CREATE SUBJECTS ====================
|Admin|
:Create subjects;

|System|
:Verify role [Admin];
if (Has role?) then (Yes)
  :Create Subject;
  |Admin|
  :Notify: Success;
else (No)
  :403 Forbidden;
endif

' ==================== ADMIN: CREATE CLASS ====================
|Admin|
:Create class;

|System|
:Verify role [Admin];
if (Has role?) then (Yes)
  :Get Subject(subjectId);
  if (Subject exists?) then (Yes)
    :Get Teacher(teacherId);
    if (Teacher exists?) then (Yes)
      :Create Class(status=OPEN,\ncapacity<=20,schedule,term);
      |Admin|
      :Notify: Success;
    else (No)
      :404 Teacher not found;
    endif
  else (No)
    :404 Subject not found;
  endif
else (No)
  :403 Forbidden;
endif

' =================== STUDENT: ENROLL ===================
|Student|
:Tries to enroll in a class;

|System|
:Verify role [Student|Admin|Teacher*policy];
if (Has role?) then (Yes)
  :Get Student(studentId);
  if (Student exists?) then (Yes)
    :Get Class(classId);
    if (Class exists?) then (Yes)
      if ([class.status = OPEN\nand seatsAvailable > 0]?) then (Yes)
        note right
          Begin TX (Enrollment + seat decrement)
        end note
        :Create Enrollment(status=ACTIVE);
        :Decrement Class.seatsAvailable;
        :Commit TX;
        |Student|
        :Enroll Success (201);
      else (No)
        |Student|
        :409 Class full/closed;
      endif
    else (No)
```

```
      :404 Class not found;
    endif
  else (No)
    :404 Student not found;
  endif
else (No)
  :403 Forbidden;
endif

' ==================== TEACHER: GRADE ====================
|Teacher|
:Tries to grade a student;

|System|
:Verify role [Teacher|Admin];
if (Has role?) then (Yes)
  :Get Enrollment(enrollmentId);
  if (Enrollment ACTIVE?) then (Yes)
    if ([0 ≤ grade ≤ 100]?) then (Yes)
      note right
        Begin TX (Grade write + audit/version)
      end note
      :Create/Update Grade(value, reason,\ngraded_by, version++);
      :Write Audit Entry;
      :Commit TX;
      |Student|
      :Notify: Grade Saved;
    else (No)
      |Teacher|
      :400 Grade out of range;
    endif
  else (No)
    :404 Enrollment not found/active;
  endif
else (No)
  :403 Forbidden;
endif

' ==================== STUDENT: VIEW PROGRESS ====================
|Student|
:View progress by term;

|System|
:Verify role [Student|Admin];
if (Has role?) then (Yes)
  :Query Read Model / DB\n(studentId, term);
  :Return progress list;
  |Student|
  :Progress shown;
else (No)
  :403 Forbidden;
endif

|Admin|
end
@enduml
```

///

```
@startuml
title School Management System — Component Diagram (Whole System)

skinparam componentStyle uml2
skinparam rectangleBackgroundColor White
skinparam rectangleBorderColor Black
skinparam noteBackgroundColor #ffffee

'==================== Front Door ====================
package "Front Door" {
  [UI / API Gateway] as APIGW
  interface IIdentityAndAccess
  APIGW -right-> IIdentityAndAccess : verifyToken()\nhasRole()
}

'==================== Identity ====================
component "Identity & Access" as IDP
IIdentityAndAccess -down-> IDP : provided

'==================== Domain Services ====================
package "Domain Services" {

  '---- Subjects
  interface ISubjectsService
  component "Subjects Management" as SUBJECTS
  SUBJECTS -up- ISubjectsService : provides

  '---- Teachers
  interface ITeacherService
  component "Teacher Management" as TEACHERS
  TEACHERS -up- ITeacherService : provides

  '---- Students
  interface IStudentService
  component "Student Registry" as STUDENTS
  STUDENTS -up- IStudentService : provides
  '---- Classes / Catalog
  interface IClassService
  component "Class Catalog" as CLASSES
  CLASSES -up- IClassService : provides

  '---- Enrollment
  interface IEnrollmentService
  component "Enrollment" as ENROLL
  ENROLL -up- IEnrollmentService : provides

  '---- Grading
  interface IGradeService
  component "Grading" as GRADING
  GRADING -up- IGradeService : provides
}

'=================== API → Domain ports ====================
APIGW -down-> ISubjectsService : manage subjects
APIGW -down-> ITeacherService : manage teachers
APIGW -down-> IStudentService : create/get students
APIGW -down-> IClassService : create/list classes
APIGW -down-> IEnrollmentService : enroll/unenroll
APIGW -down-> IGradeService : record/update grade

'=================== Inter-service required ports ====================
(IStudentService) -left- ENROLL : required
(IClassService)   -up-   ENROLL : required

(ISubjectsService) -left- CLASSES : required
(ITeacherService)  -up-   CLASSES : required

(IEnrollmentService) -left- GRADING : required

'=================== Persistence and Read Model ====================
database "Relational DB\n(owned tables per module)" as DB
rectangle "Read Model / Reports\n(optional: student progress)" as READMODEL

SUBJECTS -down-> DB : Subject tables
TEACHERS -down-> DB : Teacher tables
STUDENTS -down-> DB : Student tables
CLASSES  -down-> DB : Class tables
ENROLL   -down-> DB : Enrollment tables
GRADING  -down-> DB : Grade tables\n(version/audit)

' Optional projections
ENROLL -right-> READMODEL : publish/enrich
GRADING -right-> READMODEL : publish/enrich
APIGW -down-> READMODEL : GET /students/{id}/progress?term=
'==================== Compliance Notes =====================
note bottom of ENROLL
Transactional consistency for enrollment writes
- Create/Cancel Enrollment
- Update Class.seatsAvailable
- Guard: status=OPEN & seats>0
end note

note bottom of GRADING
Transactional consistency for grade writes
- Guard: 0 ≤ value ≤ 100
- Versioned/Audited updates
end note

note right of CLASSES
Validates references via required ports:
- ISubjectsService
- ITeacherService
end note

note right of APIGW
Role-based access checks
(Admin / Teacher / Student)
before calling domain ports
end note

@enduml
```

//

```
@startuml
title Sequence — Record Grade

actor Teacher
participant "API Gateway" as API
participant "Identity" as IDP
participant "Grading" as GR
participant "Enrollment" as EN
database "DB" as DB

Teacher -> API : POST /grades {enrollmentId, value, reason}
API -> IDP : verifyToken(); hasRole(Teacher|Admin)
IDP --> API : ok

API -> GR : CreateGrade(enrollmentId, value, reason)
GR -> EN : GetEnrollment(enrollmentId)
```

```
EN --> GR : {status: ACTIVE, studentId, classId}

alt 0 <= value <= 100
  GR -> DB : BEGIN TX
  GR -> DB : INSERT Grade(..., version=1)
  GR -> DB : INSERT AuditEntry(...)
  GR -> DB : COMMIT
  GR --> API : GradeCreated(201)
else out of range
  GR --> API : Error(400)
end

API --> Teacher : 201 Created / 400 Bad Request
@enduml

////////////////////////////////////////////////////////////////////////////////////////////////////////////

@startuml
title Sequence — Whole System (Happy Path)

actor Admin
actor Student
actor Teacher

participant "API Gateway" as API
participant "Identity & Access" as IDP
participant "Subjects" as SUBJ
participant "Teachers" as TCH
participant "Students" as STU
participant "Class Catalog" as CLS
participant "Enrollment" as ENR
participant "Grading" as GRD
database "Relational DB" as DB
collections "Read Model" as RM

' ============ Admin logs in ============
Admin -> API : POST /login {user,pass}
API -> IDP : verifyCredentials()
IDP --> API : ok + JWT
API --> Admin : 200 {JWT}

' ============ Admin creates a Subject ============
Admin -> API : POST /subjects {name}
API -> IDP : hasRole(Admin)
IDP --> API : ok
API -> SUBJ : CreateSubject(name)
SUBJ -> DB : INSERT Subject
SUBJ --> API : SubjectCreated
```

```
API --> Admin : 201 Subject

' ============ Admin creates a Class ============
Admin -> API : POST /classes {subjectId, term, teacherId, schedule, capacity<=20,
status=OPEN}
API -> IDP : hasRole(Admin)
IDP --> API : ok
API -> CLS : CreateClass(...)
CLS -> SUBJ : GetSubject(subjectId)
SUBJ --> CLS : Subject
CLS -> TCH : GetTeacher(teacherId)
TCH --> CLS : Teacher
CLS -> DB : INSERT Class (status=OPEN, capacity<=20)
CLS --> API : ClassCreated
API --> Admin : 201 Class

' ============ Student enrolls ============
Student -> API : POST /enrollments {studentId, classId}
API -> IDP : hasRole(Student|Admin|Teacher*policy)
IDP --> API : ok
API -> ENR : Enroll(studentId, classId)
ENR -> STU : GetStudent(studentId)
STU --> ENR : Student
ENR -> CLS : GetClass(classId)
CLS --> ENR : {status=OPEN, seatsAvailable>0}
ENR -> DB : BEGIN TX
ENR -> DB : INSERT Enrollment(status=ACTIVE)
ENR -> DB : UPDATE Class SET seatsAvailable=seatsAvailable-1
ENR -> DB : COMMIT
ENR --> API : EnrollmentCreated
API --> Student : 201 Enrollment

' ============ Teacher records a grade ============
Teacher -> API : POST /grades {enrollmentId, value(0..100), reason}
API -> IDP : hasRole(Teacher|Admin)
IDP --> API : ok
API -> GRD : CreateGrade(enrollmentId, value, reason)
GRD -> ENR : GetEnrollment(enrollmentId)
ENR --> GRD : {status=ACTIVE, studentId, classId}
GRD -> DB : BEGIN TX
GRD -> DB : INSERT Grade(enrollmentId, value, graded_by, version=1)
GRD -> DB : INSERT AuditEntry(...)
GRD -> DB : COMMIT
GRD --> API : GradeCreated
API --> Teacher : 201 Grade

' ============ Student views progress ============
Student -> API : GET /students/{id}/progress?term=T1
```

```
API -> RM : QueryProgress(studentId, term)
RM --> API : Progress[]
API --> Student : 200 Progress

@enduml

////////////////////////////////////////////

@startuml
title School Management System — Class Diagram (Whole System)

' ---------- Packages ----------
package "Domain Entities" {
  class Student {
    +id: UUID
    +firstName: String
    +lastName: String
    +email: String
    +status: String
    +createdAt: DateTime
    +updatedAt: DateTime
  }

  class Teacher {
    +id: UUID
    +firstName: String
    +lastName: String
    +email: String
    +createdAt: DateTime
    +updatedAt: DateTime
  }

  class Subject {
    +id: UUID
    +code: String
    +name: String
    +createdAt: DateTime
    +updatedAt: DateTime
  }

  enum ClassStatus {
    OPEN
    CLOSED
  }

  class Class {
    +id: UUID
    +subjectId: UUID
    +teacherId: UUID
    +term: String
    +schedule: String
    +status: ClassStatus
    +capacity: Int
    +seatsAvailable: Int
    +createdAt: DateTime
    +updatedAt: DateTime
  }

  enum EnrollmentStatus {
    ACTIVE
    CANCELLED
  }

  class Enrollment {
    +id: UUID
    +studentId: UUID
    +classId: UUID
    +status: EnrollmentStatus
    +createdAt: DateTime
    +updatedAt: DateTime
  }

  class Grade {
    +id: UUID
    +enrollmentId: UUID
    +value: Int
    +reason: String
    +gradedBy: UUID
    +version: Int
    +createdAt: DateTime
    +updatedAt: DateTime
  }
}

package "Service Interfaces (Ports)" {
  interface IStudentService {
    +createStudent(data): Student
    +getStudent(id: UUID): Student
    +listStudents(filter): Student[]
  }

  interface ITeacherService {
    +getTeacher(id: UUID): Teacher
    +listTeachers(filter): Teacher[]
  }
  interface ISubjectsService {
    +createSubject(data): Subject
    +getSubject(id: UUID): Subject
    +listSubjects(filter): Subject[]
  }

  interface IClassService {
    +createClass(data): Class
    +getClass(id: UUID): Class
    +listClasses(filter): Class[]
  }

  interface IEnrollmentService {
    +enroll(studentId: UUID, classId: UUID): Enrollment
    +unenroll(enrollmentId: UUID): void
    +getEnrollment(id: UUID): Enrollment
  }

  interface IGradeService {
    +createGrade(enrollmentId: UUID, value: Int, reason: String): Grade
    +updateGrade(id: UUID, value: Int, reason: String): Grade
  }
}

' ---------- Associations ----------
Student "1" -- "0..*" Enrollment : has >
Class "1" -- "0..*" Enrollment : contains >
Enrollment "1" -- "0..1" Grade : < results in
Class "many" --> "1" Subject : subject
Class "many" --> "1" Teacher : teacher

' ---------- Notes / Constraints ----------
note bottom of Class
  status must be OPEN to accept enrollments
  capacity ≤ 20; seatsAvailable decremented on enroll
end note

note bottom of Grade
  value in range 0..100
  updates are versioned/audited
  grade references an ACTIVE enrollment
end note

' ---------- Who uses what (service→entity intent) ----------
IStudentService ..> Student
ITeacherService ..> Teacher
ISubjectsService ..> Subject
IClassService ..> Class
```

```
IEnrollmentService ..> Enrollment
IGradeService ..> Grade

@enduml

////////////////////////////////////////////////////////

@startuml
title Grading Module — Class Diagram

package "Domain" {
  class Grade {
    +id: UUID
    +enrollmentId: UUID
    +value: Int        <<0..100>>
    +reason: String
    +gradedBy: UUID     <<Teacher.id>>
    +version: Int       <<>=1>>
    +createdAt: DateTime
    +updatedAt: DateTime
    --
    +applyInitial(value:Int, reason:String, by:UUID)
    +applyUpdate(value:Int, reason:String, by:UUID)
  }

  class GradeAuditEntry {
    +id: UUID
    +gradeId: UUID
    +changedAt: DateTime
    +changedBy: UUID     <<Teacher/Admin>>
    +oldValue: Int?
    +newValue: Int
    +reason: String
    +newVersion: Int
  }

  Grade "1" o-- "0..*" GradeAuditEntry : <<versioned history>>
}

package "Application (Use Cases)" {
  interface IGradeService {
    +createGrade(enrollmentId: UUID, value: Int, reason: String, gradedBy: UUID): Grade
    +updateGrade(gradeId: UUID, value: Int, reason: String, gradedBy: UUID): Grade
  }

  class GradeService implements IGradeService {
    -repo: IGradeRepository
    -enrollClient: IEnrollmentClient
    --
    +createGrade(enrollmentId: UUID, value: Int, reason: String, gradedBy: UUID): Grade
    +updateGrade(gradeId: UUID, value: Int, reason: String, gradedBy: UUID): Grade
    --
    -ensureValidRange(value:Int)
    -ensureActiveEnrollment(enrollmentId:UUID)
    -appendAudit(grade:Grade, oldValue:Int?, newValue:Int, reason:String, by:UUID)
  }
}

package "Outbound Ports (Secondary)" {
  interface IGradeRepository {
    +findById(id: UUID): Grade
    +save(grade: Grade): void             <<atomic>>
    +appendAudit(entry: GradeAuditEntry): void
  }

  interface IEnrollmentClient {
    +getEnrollment(enrollmentId: UUID): EnrollmentSnapshot
  }

  class EnrollmentSnapshot {
    +enrollmentId: UUID
    +status: String   <<ACTIVE|CANCELLED>>
    +studentId: UUID
    +classId: UUID
  }
}

' Wiring / dependencies
IGradeService ..> Grade : manages
IGradeService ..> GradeAuditEntry
GradeService ..> IGradeRepository : persist + audit
GradeService ..> IEnrollmentClient : validate context
GradeService ..> Grade

' Notes (business & NFRs)
note bottom of Grade
- Grade.value must be in [0, 100]
- version increments on every update
- gradedBy is the actor performing the action
end note

note right of GradeService
Rules enforced:
1) Role check happens at API (outside this module)
2) Must validate ACTIVE enrollment via IEnrollmentClient
3) On create/update:
   - Begin TX
   - save(Grade) + appendAudit(...)
   - Commit TX
end note

@enduml

–////////////////////////////////////////////////////////////////////////////////

@startuml
title ERD — School Management (Core Relational Model)

hide methods
skinparam linetype ortho

entity "students" as students {
  * id : uuid <<PK>>
  --
  first_name : varchar(80)
  last_name  : varchar(80)
  email      : varchar(180) <<UNIQUE>>
  status     : varchar(16)  <<ACTIVE|INACTIVE>>
  created_at : timestamptz
  updated_at : timestamptz
}

entity "teachers" as teachers {
  * id : uuid <<PK>>
  --
  first_name : varchar(80)
  last_name  : varchar(80)
  email      : varchar(180) <<UNIQUE>>
  created_at : timestamptz
  updated_at : timestamptz
}

entity "subjects" as subjects {
  * id   : uuid <<PK>>
  --
  code   : varchar(32) <<UNIQUE>>
  name   : varchar(140) <<UNIQUE>>
  created_at : timestamptz
  updated_at : timestamptz
}

entity "classes" as classes {
  * id           : uuid <<PK>>
  --
  subject_id      : uuid <<FK subjects.id>>
  teacher_id      : uuid <<FK teachers.id>>
  term            : varchar(32)
  schedule        : varchar(140)
  status          : varchar(10) <<OPEN|CLOSED>>
  capacity        : int         <<CHECK: 1..20>>
  seats_available : int         <<CHECK: 0..capacity>>
  created_at      : timestamptz
  updated_at      : timestamptz
  --
  <<UNIQUE(subject_id, term, schedule)>>
}

entity "enrollments" as enrollments {
  * id         : uuid <<PK>>
  --
  student_id   : uuid <<FK students.id>>
  class_id     : uuid <<FK classes.id>>
  status       : varchar(12) <<ACTIVE|CANCELLED>>
  created_at   : timestamptz
  updated_at   : timestamptz
  --
  <<UNQ PARTIAL: (student_id,class_id) WHERE status='ACTIVE'>>
}

entity "grades" as grades {
  * id           : uuid <<PK>>
  --
  enrollment_id   : uuid <<FK enrollments.id>> <<UNIQUE>>
  value          : int  <<CHECK: 0..100>>
  reason         : varchar(255)
  graded_by      : uuid <<FK teachers.id>>
  version        : int  <<>=1>>
  created_at     : timestamptz
  updated_at     : timestamptz
}

entity "grade_audit" as grade_audit {
  * id          : uuid <<PK>>
  --
  grade_id      : uuid <<FK grades.id>>
  changed_at    : timestamptz
  changed_by    : uuid <<FK teachers.id>>
  old_value     : int?  <<CHECK: 0..100 or NULL>>
  new_value     : int   <<CHECK: 0..100>>
  reason        : varchar(255)
  new_version   : int   <<>=1>>
}
```

```
' -------------------- Relationships (crow's foot-ish) --------------------
students   ||--o{ enrollments : has
classes    ||--o{ enrollments : contains
enrollments ||--|| grades     : current_grade
grades     ||--o{ grade_audit : history
subjects   ||--o{ classes     : categorizes
teachers   ||--o{ classes     : teaches
teachers   ||--o{ grades      : graded_by

@enduml

//////////////////////////////////////////////////////////////////////////////////////////
```