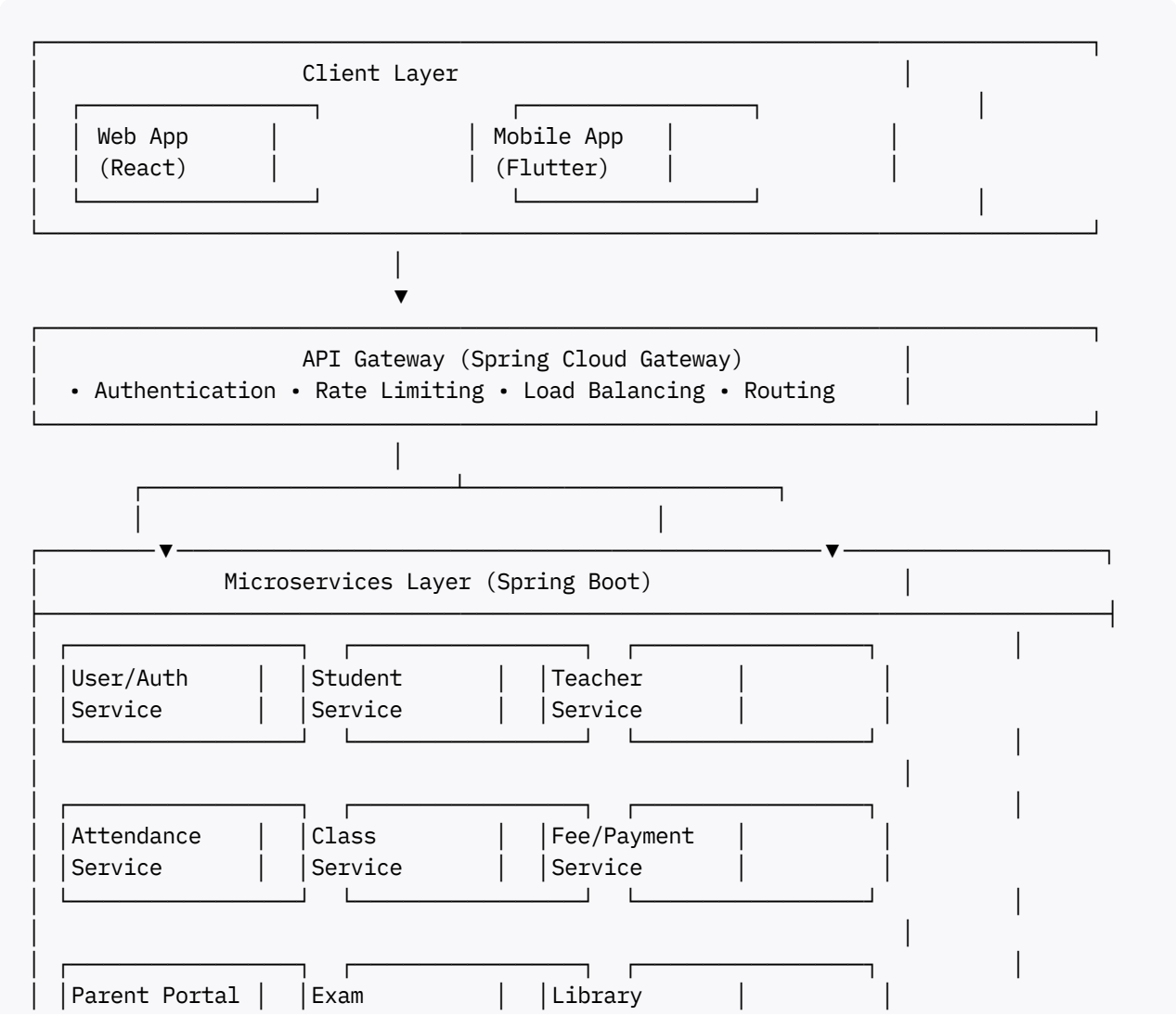
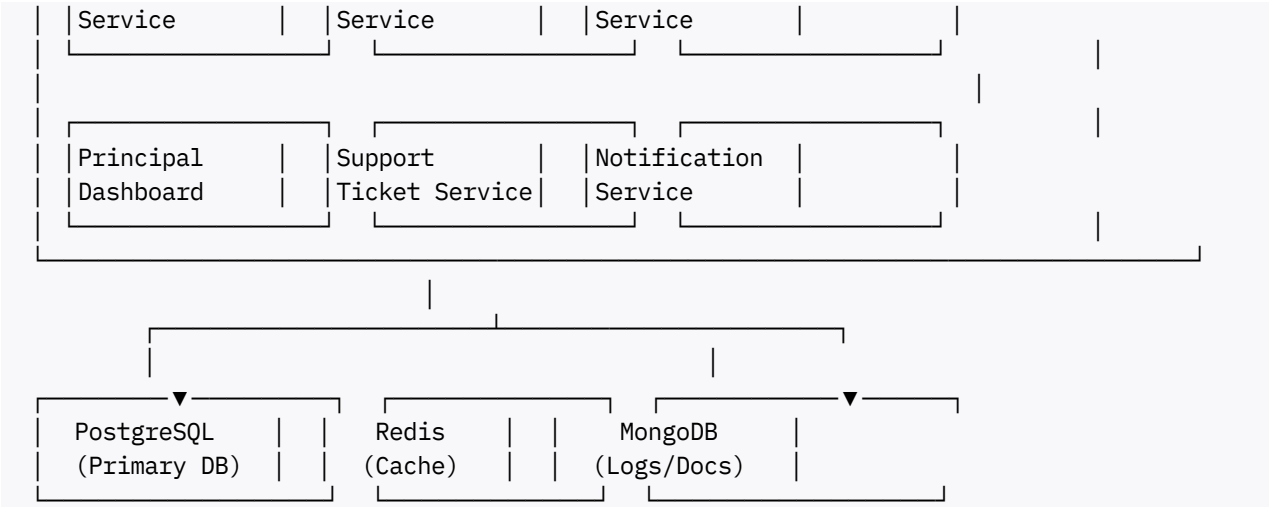
perplexity

# Generate a complete HLD and LLD diagrams for each and every services. Also provide each and every API requests required and describe their their functionalities. Provide all create table statements in PostgreSQL for each and every table required for each services.

I'll provide comprehensive HLD and LLD diagrams, all API endpoints with descriptions, and complete PostgreSQL table schemas for each service in the School Management System.

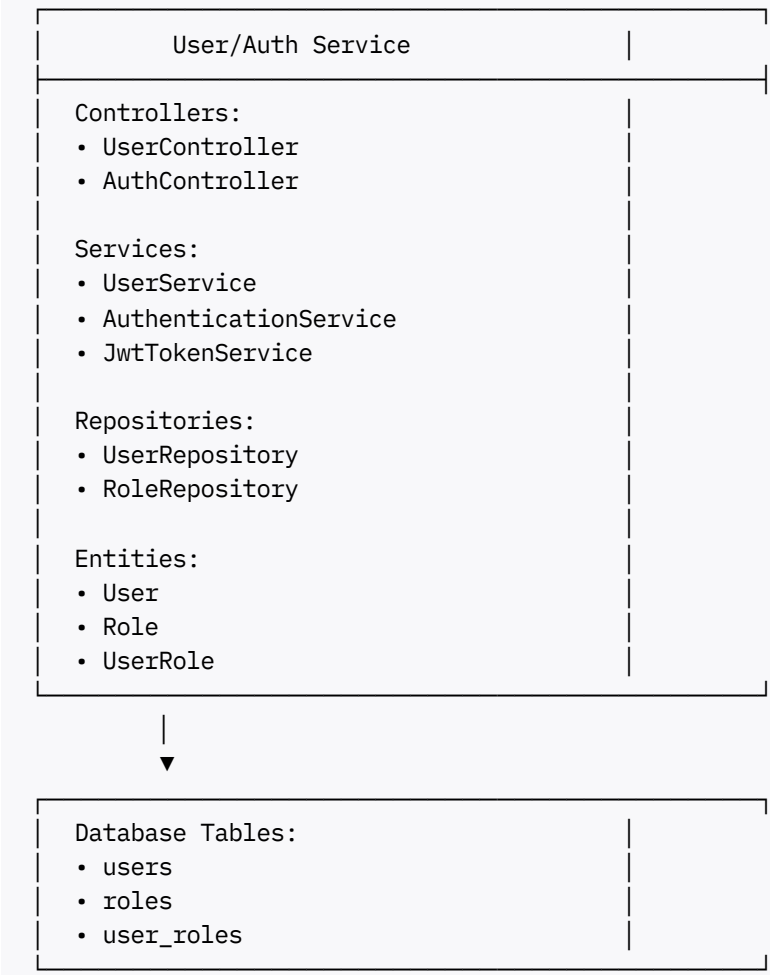
### Complete High-Level Design (HLD) Diagram






## Low-Level Design (LLD) Diagrams for Each Service

### 1. User/Auth Service LLD

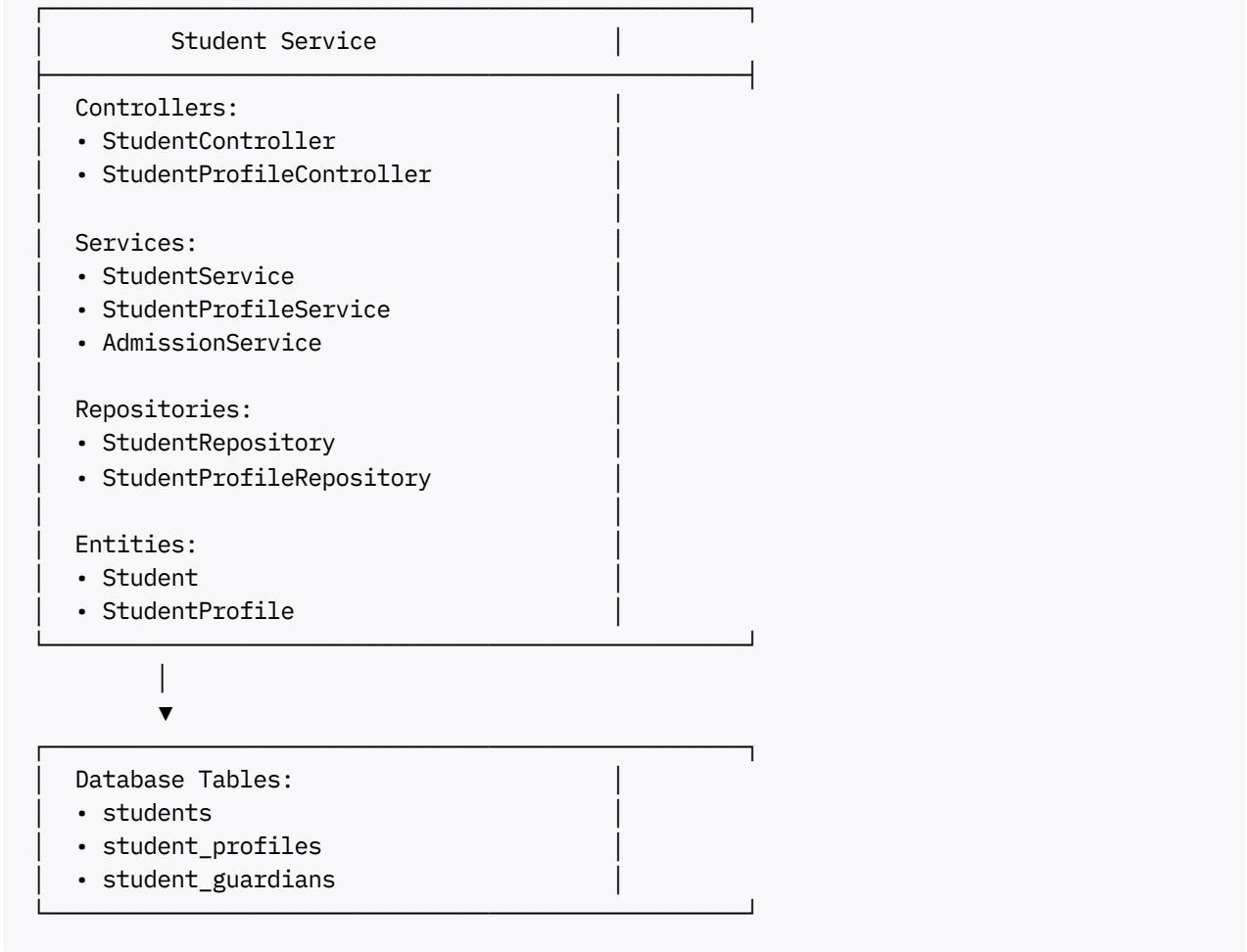



**Class Diagram:**

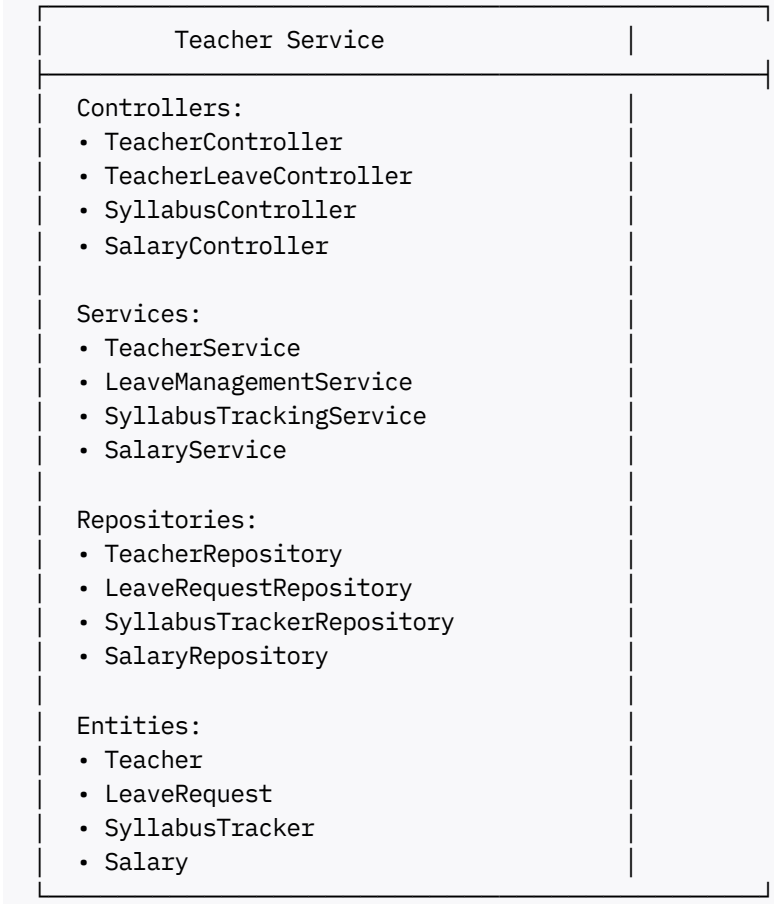
```
User
├── user_id: Long
├── username: String
├── password_hash: String
├── email: String
├── phone: String
├── status: String
├── created_at: Timestamp
└── roles: List<Role>

Role
├── role_id: Long
├── role_name: String
└── users: List<User>
```

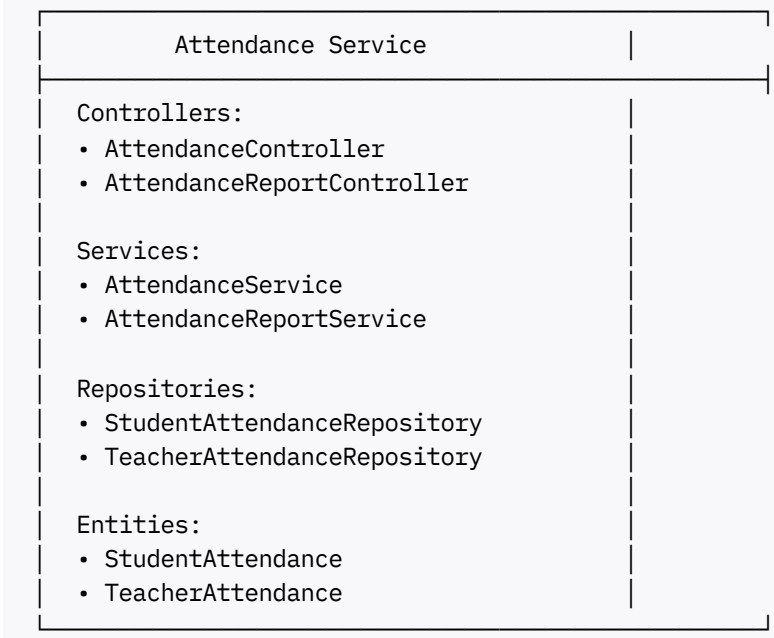
## 2. Student Service LLD



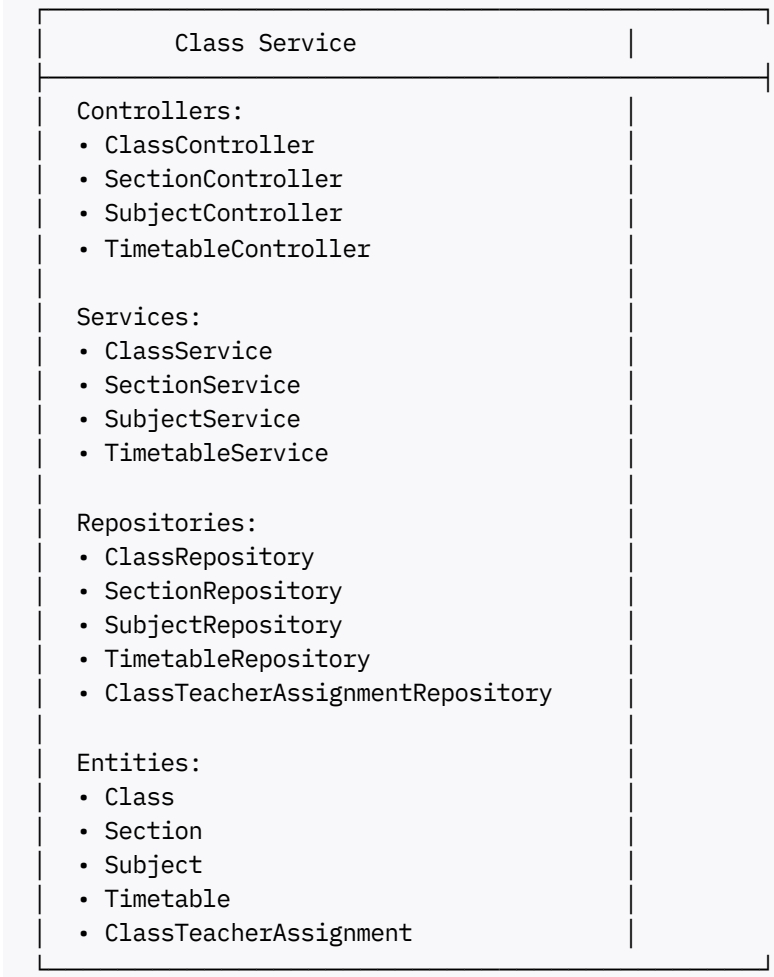
### 3. Teacher Service LLD



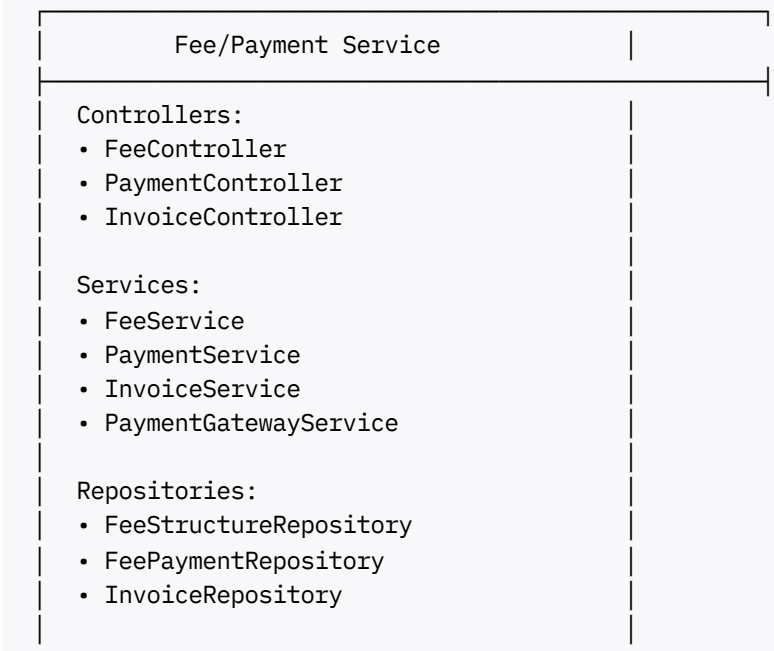

### 4. Attendance Service LLD



## 5. Class Service LLD




## 6. Fee/Payment Service LLD



```
│  Entities:                                  │
│  • FeeStructure                             │
│  • FeePayment                               │
│  • Invoice                                  │
└─────────────────────────────────────────────┘
```

### 7. Parent Portal Service LLD

```
┌─────────────────────────────────────────────┐
│         Parent Portal Service               │
├─────────────────────────────────────────────┤
│  Controllers:                               │
│  • ParentController                         │
│  • ParentDashboardController                │
│                                             │
│  Services:                                  │
│  • ParentService                            │
│  • ChildInfoService                         │
│                                             │
│  Repositories:                              │
│  • ParentRepository                         │
│  • ParentStudentMappingRepository           │
│                                             │
│  Entities:                                  │
│  • Parent                                   │
│  • ParentStudentMapping                     │
└─────────────────────────────────────────────┘
```

### 8. Exam Service LLD

```
┌─────────────────────────────────────────────┐
│         Exam Service                        │
├─────────────────────────────────────────────┤
│  Controllers:                               │
│  • ExamController                           │
│  • ExamScheduleController                   │
│  • MarksController                          │
│  • ReportCardController                     │
│                                             │
│  Services:                                  │
│  • ExamService                              │
│  • ExamScheduleService                      │
│  • MarksEntryService                        │
│  • ReportCardService                        │
│                                             │
│  Repositories:                              │
│  • ExamRepository                           │
│  • ExamScheduleRepository                   │
│  • ExamResultRepository                     │
│  • ReportCardRepository                     │
│                                             │
```

```
│  Entities:                                  │
│  • Exam                                     │
│  • ExamSchedule                             │
│  • ExamResult                               │
│  • ReportCard                               │
└─────────────────────────────────────────────┘
```

### 9. Library Service LLD

```
┌─────────────────────────────────────────────┐
│          Library Service                    │
├─────────────────────────────────────────────┤
│  Controllers:                               │
│  • BookController                           │
│  • LibraryIssueController                   │
│  • FineController                           │
│                                             │
│  Services:                                  │
│  • BookService                              │
│  • LibraryIssueService                      │
│  • FineCalculationService                   │
│                                             │
│  Repositories:                              │
│  • BookRepository                           │
│  • LibraryIssueRepository                   │
│  • LibraryFineRepository                    │
│                                             │
│  Entities:                                  │
│  • Book                                     │
│  • LibraryIssue                             │
│  • LibraryFine                              │
└─────────────────────────────────────────────┘
```

### 10. Principal Dashboard Service LLD

```
┌─────────────────────────────────────────────┐
│      Principal Dashboard Service            │
├─────────────────────────────────────────────┤
│  Controllers:                               │
│  • PrincipalDashboardController             │
│  • AnalyticsController                      │
│  • AnnouncementController                   │
│                                             │
│  Services:                                  │
│  • DashboardService                         │
│  • AnalyticsService                         │
│  • AnnouncementService                      │
│                                             │
│  Repositories:                              │
│  • AnnouncementRepository                   │
│                                             │
```

```
│  Entities:                                │
│  • Announcement                           │
└──────────────────────────────────────────┘
```

### 11. Support Ticket Service LLD

```
┌──────────────────────────────────────────┐
│      Support Ticket Service              │
├──────────────────────────────────────────┤
│  Controllers:                            │
│  • SupportTicketController               │
│  • AdminTicketController                 │
│                                          │
│  Services:                               │
│  • TicketService                         │
│  • TicketAssignmentService               │
│                                          │
│  Repositories:                           │
│  • SupportTicketRepository               │
│  • TicketCommentRepository               │
│                                          │
│  Entities:                               │
│  • SupportTicket                         │
│  • TicketComment                         │
└──────────────────────────────────────────┘
```

### 12. Notification Service LLD

```
┌──────────────────────────────────────────┐
│      Notification Service                │
├──────────────────────────────────────────┤
│  Controllers:                            │
│  • NotificationController                │
│                                          │
│  Services:                               │
│  • NotificationService                   │
│  • EmailService                          │
│  • SMSService                            │
│  • PushNotificationService               │
│                                          │
│  Repositories:                           │
│  • NotificationRepository                │
│                                          │
│  Entities:                               │
│  • Notification                          │
└──────────────────────────────────────────┘
```

## Complete API Endpoints for All Services

### 1. User/Auth Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/auth/register` | Register new user |
| POST | `/api/auth/login` | User login and JWT token generation |
| POST | `/api/auth/logout` | User logout |
| POST | `/api/auth/refresh-token` | Refresh JWT token |
| POST | `/api/auth/forgot-password` | Initiate password reset |
| POST | `/api/auth/reset-password` | Complete password reset |
| GET | `/api/users` | Get all users (Admin only) |
| GET | `/api/users/{id}` | Get user by ID |
| PUT | `/api/users/{id}` | Update user profile |
| DELETE | `/api/users/{id}` | Delete user (Admin only) |
| GET | `/api/users/{id}/roles` | Get user roles |
| POST | `/api/users/{id}/roles` | Assign role to user |
| DELETE | `/api/users/{id}/roles/{roleId}` | Remove role from user |

### 2. Student Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/students` | Create new student admission |
| GET | `/api/students` | Get all students with pagination |
| GET | `/api/students/{id}` | Get student by ID |
| PUT | `/api/students/{id}` | Update student information |
| DELETE | `/api/students/{id}` | Delete student record |
| GET | `/api/students/{id}/profile` | Get detailed student profile |
| PUT | `/api/students/{id}/profile` | Update student profile |
| GET | `/api/students/class/{classId}` | Get all students in a class |
| GET | `/api/students/search` | Search students by name/roll |
| POST | `/api/students/{id}/promote` | Promote student to next class |
| GET | `/api/students/{id}/guardians` | Get student guardians |
| POST | `/api/students/{id}/guardians` | Add guardian for student |

## 3. Teacher Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/teachers` | Create new teacher |
| GET | `/api/teachers` | Get all teachers |
| GET | `/api/teachers/{id}` | Get teacher by ID |
| PUT | `/api/teachers/{id}` | Update teacher information |
| DELETE | `/api/teachers/{id}` | Delete teacher record |
| GET | `/api/teachers/{id}/schedule` | Get teacher schedule |
| POST | `/api/teachers/{id}/leave` | Apply for leave |
| GET | `/api/teachers/{id}/leave` | Get teacher leave history |
| PUT | `/api/teachers/leave/{leaveId}/approve` | Approve leave request |
| PUT | `/api/teachers/leave/{leaveId}/reject` | Reject leave request |
| GET | `/api/teachers/{id}/syllabus` | Get syllabus completion status |
| PUT | `/api/teachers/{id}/syllabus` | Update syllabus progress |
| GET | `/api/teachers/{id}/salary` | Get salary details |
| POST | `/api/teachers/{id}/salary` | Process salary payment |
| GET | `/api/teachers/{id}/payslips` | Get payslip history |

## 4. Attendance Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/attendance/student` | Mark student attendance |
| POST | `/api/attendance/student/bulk` | Mark attendance for multiple students |
| GET | `/api/attendance/student/{studentId}` | Get student attendance records |
| GET | `/api/attendance/student/{studentId}/summary` | Get attendance summary/percentage |
| GET | `/api/attendance/class/{classId}` | Get class attendance for date |
| POST | `/api/attendance/teacher` | Mark teacher attendance |
| GET | `/api/attendance/teacher/{teacherId}` | Get teacher attendance records |
| GET | `/api/attendance/report/daily` | Generate daily attendance report |

| HTTP Method | Endpoint | Description |
|---|---|---|
| GET | `/api/attendance/report/monthly` | Generate monthly attendance report |
| PUT | `/api/attendance/{attendanceId}` | Update attendance record |
| GET | `/api/attendance/defaulters` | Get list of attendance defaulters |

## 5. Class Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/classes` | Create new class |
| GET | `/api/classes` | Get all classes |
| GET | `/api/classes/{id}` | Get class by ID |
| PUT | `/api/classes/{id}` | Update class information |
| DELETE | `/api/classes/{id}` | Delete class |
| POST | `/api/classes/{classId}/sections` | Create section in class |
| GET | `/api/classes/{classId}/sections` | Get all sections in class |
| PUT | `/api/sections/{sectionId}` | Update section |
| DELETE | `/api/sections/{sectionId}` | Delete section |
| POST | `/api/subjects` | Create new subject |
| GET | `/api/subjects` | Get all subjects |
| GET | `/api/subjects/class/{classId}` | Get subjects for class |
| PUT | `/api/subjects/{id}` | Update subject |
| DELETE | `/api/subjects/{id}` | Delete subject |
| POST | `/api/classes/{classId}/assign-teacher` | Assign teacher to class/subject |
| GET | `/api/classes/{classId}/teachers` | Get teachers assigned to class |
| DELETE | `/api/classes/assignment/{assignmentId}` | Remove teacher assignment |
| POST | `/api/timetable` | Create timetable |
| GET | `/api/timetable/class/{classId}` | Get timetable for class |
| PUT | `/api/timetable/{id}` | Update timetable |

## 6. Fee/Payment Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/fees/structure` | Create fee structure |
| GET | `/api/fees/structure` | Get all fee structures |
| GET | `/api/fees/structure/{id}` | Get fee structure by ID |
| PUT | `/api/fees/structure/{id}` | Update fee structure |
| POST | `/api/fees/invoice` | Generate fee invoice |
| GET | `/api/fees/invoice/{invoiceId}` | Get invoice details |
| GET | `/api/fees/student/{studentId}/invoices` | Get student invoices |
| POST | `/api/fees/pay` | Process fee payment |
| GET | `/api/fees/payment/{paymentId}` | Get payment details |
| GET | `/api/fees/student/{studentId}/payments` | Get student payment history |
| GET | `/api/fees/student/{studentId}/due` | Get due amount for student |
| GET | `/api/fees/defaulters` | Get list of fee defaulters |
| POST | `/api/fees/reminder` | Send fee payment reminder |
| GET | `/api/fees/reports/collection` | Get fee collection report |

## 7. Parent Portal Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| GET | `/api/parent/{parentId}/children` | Get all children of parent |
| GET | `/api/parent/{parentId}/dashboard` | Get parent dashboard summary |
| GET | `/api/parent/student/{studentId}/attendance` | View child attendance |
| GET | `/api/parent/student/{studentId}/performance` | View child academic performance |
| GET | `/api/parent/student/{studentId}/exams` | View child exam schedule |
| GET | `/api/parent/student/{studentId}/fees` | View child fee status |
| GET | `/api/parent/student/{studentId}/homework` | View child homework |
| GET | `/api/parent/announcements` | View school announcements |
| POST | `/api/parent/feedback` | Submit parent feedback |
| GET | `/api/parent/student/{studentId}/timetable` | View child timetable |

## 8. Exam Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/exams` | Create new exam |
| GET | `/api/exams` | Get all exams |
| GET | `/api/exams/{id}` | Get exam by ID |
| PUT | `/api/exams/{id}` | Update exam details |
| DELETE | `/api/exams/{id}` | Delete exam |
| POST | `/api/exams/{examId}/schedule` | Create exam schedule |
| GET | `/api/exams/{examId}/schedule` | Get exam schedule |
| PUT | `/api/exams/schedule/{scheduleId}` | Update exam schedule |
| POST | `/api/exams/marks` | Enter exam marks |
| PUT | `/api/exams/marks/{resultId}` | Update exam marks |
| GET | `/api/exams/{examId}/results` | Get all exam results |
| GET | `/api/exams/student/{studentId}/results` | Get student exam results |
| POST | `/api/exams/{examId}/report-card` | Generate report cards |
| GET | `/api/exams/report-card/{studentId}` | Get student report card |
| GET | `/api/exams/class/{classId}` | Get exams for class |
| POST | `/api/exams/{examId}/publish` | Publish exam results |

## 9. Library Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/library/books` | Add new book to library |
| GET | `/api/library/books` | Get all books |
| GET | `/api/library/books/{id}` | Get book by ID |
| PUT | `/api/library/books/{id}` | Update book information |
| DELETE | `/api/library/books/{id}` | Delete book |
| GET | `/api/library/books/search` | Search books by title/author/ISBN |
| POST | `/api/library/issue` | Issue book to student |
| PUT | `/api/library/return/{issueId}` | Return book |
| GET | `/api/library/issues/student/{studentId}` | Get books issued to student |
| GET | `/api/library/issues/active` | Get all active book issues |

| HTTP Method | Endpoint | Description |
|---|---|---|
| GET | `/api/library/issues/overdue` | Get overdue book returns |
| POST | `/api/library/fine/{issueId}` | Calculate and apply fine |
| GET | `/api/library/fine/student/{studentId}` | Get student fine details |
| POST | `/api/library/fine/pay` | Pay library fine |
| GET | `/api/library/books/available` | Get available books |

## 10. Principal Dashboard Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| GET | `/api/principal/dashboard` | Get principal dashboard overview |
| GET | `/api/principal/analytics/attendance` | Get school-wide attendance analytics |
| GET | `/api/principal/analytics/fees` | Get fee collection analytics |
| GET | `/api/principal/analytics/syllabus` | Get syllabus completion analytics |
| GET | `/api/principal/analytics/teacher-leave` | Get teacher leave analytics |
| GET | `/api/principal/analytics/performance` | Get academic performance analytics |
| POST | `/api/principal/announcements` | Create school announcement |
| GET | `/api/principal/announcements` | Get all announcements |
| PUT | `/api/principal/announcements/{id}` | Update announcement |
| DELETE | `/api/principal/announcements/{id}` | Delete announcement |
| POST | `/api/principal/broadcast` | Broadcast message to all users |
| GET | `/api/principal/reports/attendance` | Generate attendance reports |
| GET | `/api/principal/reports/financial` | Generate financial reports |

## 11. Support Ticket Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/support/tickets` | Create new support ticket |
| GET | `/api/support/tickets` | Get all tickets (Admin view) |
| GET | `/api/support/tickets/{id}` | Get ticket by ID |
| GET | `/api/support/tickets/user/{userId}` | Get tickets by user |
| PUT | `/api/support/tickets/{id}` | Update ticket |
| PATCH | `/api/support/tickets/{id}/status` | Update ticket status |

| HTTP Method | Endpoint | Description |
|---|---|---|
| PATCH | `/api/support/tickets/{id}/assign` | Assign ticket to admin |
| POST | `/api/support/tickets/{id}/comments` | Add comment to ticket |
| GET | `/api/support/tickets/{id}/comments` | Get ticket comments |
| GET | `/api/support/tickets/open` | Get all open tickets |
| GET | `/api/support/tickets/assigned/{adminId}` | Get tickets assigned to admin |
| PUT | `/api/support/tickets/{id}/resolve` | Resolve ticket |
| PUT | `/api/support/tickets/{id}/close` | Close ticket |

## 12. Notification Service APIs

| HTTP Method | Endpoint | Description |
|---|---|---|
| POST | `/api/notifications/send` | Send notification to user |
| POST | `/api/notifications/broadcast` | Broadcast notification to role/group |
| GET | `/api/notifications/user/{userId}` | Get user notifications |
| PUT | `/api/notifications/{id}/read` | Mark notification as read |
| DELETE | `/api/notifications/{id}` | Delete notification |
| POST | `/api/notifications/email` | Send email notification |
| POST | `/api/notifications/sms` | Send SMS notification |
| POST | `/api/notifications/push` | Send push notification |
| GET | `/api/notifications/user/{userId}/unread` | Get unread notifications count |

## Complete PostgreSQL CREATE TABLE Statements

### User/Auth Service Tables

```
-- Users table
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    email VARCHAR(200) UNIQUE NOT NULL,
    phone VARCHAR(15),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    status VARCHAR(10) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE','INACTIVE','SUSPENDED'
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP,
    last_login TIMESTAMP
```

```
);

-- Roles table
CREATE TABLE roles (
    role_id SERIAL PRIMARY KEY,
    role_name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);

-- User-Role mapping table
CREATE TABLE user_roles (
    user_role_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    role_id INTEGER NOT NULL REFERENCES roles(role_id) ON DELETE CASCADE,
    assigned_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(user_id, role_id)
);

-- Password reset tokens
CREATE TABLE password_reset_tokens (
    token_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    token VARCHAR(255) UNIQUE NOT NULL,
    expiry_date TIMESTAMP NOT NULL,
    used BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Refresh tokens for JWT
CREATE TABLE refresh_tokens (
    token_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    token VARCHAR(500) UNIQUE NOT NULL,
    expiry_date TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Audit log
CREATE TABLE audit_logs (
    log_id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(user_id),
    action VARCHAR(100) NOT NULL,
    entity_type VARCHAR(50),
    entity_id INTEGER,
    ip_address VARCHAR(45),
    created_at TIMESTAMP DEFAULT NOW()
);
```

### Student Service Tables

```
-- Students table
CREATE TABLE students (
    student_id SERIAL PRIMARY KEY,
    user_id INTEGER UNIQUE NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    roll_number VARCHAR(20) UNIQUE NOT NULL,
    admission_number VARCHAR(50) UNIQUE NOT NULL,
    class_id INTEGER NOT NULL,
    section_id INTEGER,
    admission_date DATE NOT NULL,
    dob DATE NOT NULL,
    gender VARCHAR(10) CHECK (gender IN ('MALE','FEMALE','OTHER')),
    blood_group VARCHAR(5),
    address TEXT,
    city VARCHAR(100),
    state VARCHAR(100),
    pincode VARCHAR(10),
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE','GRADUATED','TRANSFERR
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Student profiles (additional details)
CREATE TABLE student_profiles (
    profile_id SERIAL PRIMARY KEY,
    student_id INTEGER UNIQUE NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    photo_url VARCHAR(500),
    birth_certificate_url VARCHAR(500),
    aadhar_number VARCHAR(12),
    previous_school VARCHAR(200),
    medical_conditions TEXT,
    allergies TEXT,
    emergency_contact VARCHAR(15),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Student guardians/parents
CREATE TABLE student_guardians (
    guardian_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    user_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    relation VARCHAR(20) NOT NULL CHECK (relation IN ('FATHER','MOTHER','GUARDIAN','OTHER
    is_primary BOOLEAN DEFAULT FALSE,
    occupation VARCHAR(100),
    annual_income DECIMAL(12,2),
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(student_id, user_id)
);

-- Student documents
CREATE TABLE student_documents (
    document_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    document_type VARCHAR(50) NOT NULL,
```

```
    document_name VARCHAR(200) NOT NULL,
    document_url VARCHAR(500) NOT NULL,
    uploaded_at TIMESTAMP DEFAULT NOW()
);
```

### Teacher Service Tables

```
-- Teachers table
CREATE TABLE teachers (
    teacher_id SERIAL PRIMARY KEY,
    user_id INTEGER UNIQUE NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    employee_id VARCHAR(50) UNIQUE NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(200) UNIQUE NOT NULL,
    phone VARCHAR(15) NOT NULL,
    dob DATE,
    gender VARCHAR(10) CHECK (gender IN ('MALE','FEMALE','OTHER')),
    qualification VARCHAR(200),
    specialization VARCHAR(200),
    experience_years INTEGER,
    joining_date DATE NOT NULL,
    address TEXT,
    city VARCHAR(100),
    state VARCHAR(100),
    pincode VARCHAR(10),
    status VARCHAR(20) DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE','ON_LEAVE','RESIGNED',
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Teacher leave requests
CREATE TABLE teacher_leave_requests (
    leave_id SERIAL PRIMARY KEY,
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    leave_type VARCHAR(20) NOT NULL CHECK (leave_type IN ('SICK','CASUAL','EARNED','MATER
    reason TEXT NOT NULL,
    status VARCHAR(20) DEFAULT 'PENDING' CHECK (status IN ('PENDING','APPROVED','REJECTED
    approved_by_principal_id INTEGER REFERENCES users(user_id),
    approval_date TIMESTAMP,
    rejection_reason TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Teacher leave balance
CREATE TABLE teacher_leave_balance (
    balance_id SERIAL PRIMARY KEY,
    teacher_id INTEGER UNIQUE NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    sick_leave INTEGER DEFAULT 0,
    casual_leave INTEGER DEFAULT 0,
    earned_leave INTEGER DEFAULT 0,
```

```
    year INTEGER NOT NULL,
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Syllabus tracker
CREATE TABLE syllabus_tracker (
    tracker_id SERIAL PRIMARY KEY,
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    class_id INTEGER NOT NULL,
    subject_id INTEGER NOT NULL,
    topic VARCHAR(200) NOT NULL,
    planned_date DATE,
    completed_date DATE,
    status VARCHAR(20) DEFAULT 'PENDING' CHECK (status IN ('PENDING','IN_PROGRESS','COMPL
    remarks TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Teacher salary
CREATE TABLE teacher_salary (
    salary_id SERIAL PRIMARY KEY,
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    basic_salary DECIMAL(10,2) NOT NULL,
    hra DECIMAL(10,2),
    da DECIMAL(10,2),
    other_allowances DECIMAL(10,2),
    gross_salary DECIMAL(10,2) NOT NULL,
    pf_deduction DECIMAL(10,2),
    tax_deduction DECIMAL(10,2),
    other_deductions DECIMAL(10,2),
    net_salary DECIMAL(10,2) NOT NULL,
    month INTEGER NOT NULL CHECK (month BETWEEN 1 AND 12),
    year INTEGER NOT NULL,
    payment_date DATE,
    payment_status VARCHAR(20) DEFAULT 'PENDING' CHECK (payment_status IN ('PENDING','PAI
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(teacher_id, month, year)
);

-- Teacher attendance
CREATE TABLE teacher_attendance (
    attendance_id SERIAL PRIMARY KEY,
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    date DATE NOT NULL,
    status VARCHAR(10) NOT NULL CHECK (status IN ('PRESENT','ABSENT','LEAVE','HALF_DAY'))
    check_in_time TIME,
    check_out_time TIME,
    remarks TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(teacher_id, date)
);
```

### Attendance Service Tables

```
-- Student attendance
CREATE TABLE student_attendance (
    attendance_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    class_id INTEGER NOT NULL,
    section_id INTEGER,
    date DATE NOT NULL,
    status VARCHAR(10) NOT NULL CHECK (status IN ('PRESENT','ABSENT','LEAVE','HALF_DAY','
    subject_id INTEGER,
    marked_by_teacher_id INTEGER REFERENCES teachers(teacher_id),
    remarks TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP,
    UNIQUE(student_id, date, subject_id)
);

-- Attendance summary (for quick reporting)
CREATE TABLE student_attendance_summary (
    summary_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    month INTEGER NOT NULL CHECK (month BETWEEN 1 AND 12),
    year INTEGER NOT NULL,
    total_days INTEGER DEFAULT 0,
    present_days INTEGER DEFAULT 0,
    absent_days INTEGER DEFAULT 0,
    leave_days INTEGER DEFAULT 0,
    attendance_percentage DECIMAL(5,2),
    updated_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(student_id, month, year)
);
```

### Class Service Tables

```
-- Classes table
CREATE TABLE classes (
    class_id SERIAL PRIMARY KEY,
    class_name VARCHAR(50) NOT NULL,
    class_numeric INTEGER NOT NULL,
    academic_year VARCHAR(10) NOT NULL,
    class_teacher_id INTEGER REFERENCES teachers(teacher_id),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP,
    UNIQUE(class_name, academic_year)
);

-- Sections table
CREATE TABLE sections (
    section_id SERIAL PRIMARY KEY,
    class_id INTEGER NOT NULL REFERENCES classes(class_id) ON DELETE CASCADE,
    section_name VARCHAR(20) NOT NULL,
    capacity INTEGER DEFAULT 40,
```

```
    room_number VARCHAR(20),
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(class_id, section_name)
);

-- Subjects table
CREATE TABLE subjects (
    subject_id SERIAL PRIMARY KEY,
    subject_name VARCHAR(100) NOT NULL,
    subject_code VARCHAR(20) UNIQUE NOT NULL,
    class_id INTEGER REFERENCES classes(class_id) ON DELETE CASCADE,
    subject_type VARCHAR(20) CHECK (subject_type IN ('THEORY','PRACTICAL','BOTH')),
    max_marks INTEGER,
    pass_marks INTEGER,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Class-Teacher-Subject assignment
CREATE TABLE class_teacher_assignments (
    assignment_id SERIAL PRIMARY KEY,
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    class_id INTEGER NOT NULL REFERENCES classes(class_id) ON DELETE CASCADE,
    section_id INTEGER REFERENCES sections(section_id) ON DELETE CASCADE,
    subject_id INTEGER NOT NULL REFERENCES subjects(subject_id) ON DELETE CASCADE,
    academic_year VARCHAR(10) NOT NULL,
    assigned_date DATE DEFAULT NOW(),
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(teacher_id, class_id, section_id, subject_id, academic_year)
);

-- Timetable
CREATE TABLE timetable (
    timetable_id SERIAL PRIMARY KEY,
    class_id INTEGER NOT NULL REFERENCES classes(class_id) ON DELETE CASCADE,
    section_id INTEGER REFERENCES sections(section_id) ON DELETE CASCADE,
    subject_id INTEGER NOT NULL REFERENCES subjects(subject_id) ON DELETE CASCADE,
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    day_of_week INTEGER NOT NULL CHECK (day_of_week BETWEEN 1 AND 7),
    period_number INTEGER NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    room_number VARCHAR(20),
    academic_year VARCHAR(10) NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(class_id, section_id, day_of_week, period_number, academic_year)
);
```

### Fee/Payment Service Tables

```
-- Fee structure
CREATE TABLE fee_structure (
    fee_structure_id SERIAL PRIMARY KEY,
    class_id INTEGER NOT NULL REFERENCES classes(class_id) ON DELETE CASCADE,
    fee_type VARCHAR(50) NOT NULL CHECK (fee_type IN ('TUITION','TRANSPORT','EXAM','LIBRA
```

```
    amount DECIMAL(10,2) NOT NULL,
    frequency VARCHAR(20) NOT NULL CHECK (frequency IN ('MONTHLY','QUARTERLY','HALF_YEARL
    academic_year VARCHAR(10) NOT NULL,
    due_date INTEGER,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Fee invoices
CREATE TABLE fee_invoices (
    invoice_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    invoice_number VARCHAR(50) UNIQUE NOT NULL,
    invoice_date DATE NOT NULL,
    due_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    discount DECIMAL(10,2) DEFAULT 0,
    tax DECIMAL(10,2) DEFAULT 0,
    net_amount DECIMAL(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'PENDING' CHECK (status IN ('PENDING','PARTIALLY_PAID','PA
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Fee invoice items (breakdown of fees in invoice)
CREATE TABLE fee_invoice_items (
    item_id SERIAL PRIMARY KEY,
    invoice_id INTEGER NOT NULL REFERENCES fee_invoices(invoice_id) ON DELETE CASCADE,
    fee_structure_id INTEGER NOT NULL REFERENCES fee_structure(fee_structure_id),
    description VARCHAR(200),
    amount DECIMAL(10,2) NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Fee payments
CREATE TABLE fee_payments (
    payment_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    invoice_id INTEGER REFERENCES fee_invoices(invoice_id),
    receipt_number VARCHAR(50) UNIQUE NOT NULL,
    payment_date DATE NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    payment_mode VARCHAR(20) NOT NULL CHECK (payment_mode IN ('CASH','CHEQUE','ONLINE','C
    transaction_ref VARCHAR(100),
    status VARCHAR(20) DEFAULT 'SUCCESS' CHECK (status IN ('SUCCESS','FAILED','PENDING','
    remarks TEXT,
    received_by_user_id INTEGER REFERENCES users(user_id),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Fee defaulters tracking
CREATE TABLE fee_defaulters (
    defaulter_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    outstanding_amount DECIMAL(10,2) NOT NULL,
```

```
    last_payment_date DATE,
    reminder_sent_count INTEGER DEFAULT 0,
    last_reminder_date DATE,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);
```

### Parent Portal Service Tables

```
-- Parent-Student mapping (already covered in student_guardians)
-- Additional feedback table
CREATE TABLE parent_feedback (
    feedback_id SERIAL PRIMARY KEY,
    parent_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    student_id INTEGER REFERENCES students(student_id) ON DELETE CASCADE,
    feedback_type VARCHAR(50) NOT NULL,
    subject VARCHAR(200),
    message TEXT NOT NULL,
    status VARCHAR(20) DEFAULT 'SUBMITTED' CHECK (status IN ('SUBMITTED','REVIEWED','RESP
    response TEXT,
    responded_by INTEGER REFERENCES users(user_id),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Parent-teacher communication
CREATE TABLE parent_teacher_communication (
    communication_id SERIAL PRIMARY KEY,
    parent_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    subject VARCHAR(200),
    message TEXT NOT NULL,
    sent_by VARCHAR(10) CHECK (sent_by IN ('PARENT','TEACHER')),
    read_status BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);
```

### Exam Service Tables

```
-- Exams
CREATE TABLE exams (
    exam_id SERIAL PRIMARY KEY,
    exam_name VARCHAR(100) NOT NULL,
    exam_type VARCHAR(50) NOT NULL CHECK (exam_type IN ('UNIT_TEST','MONTHLY','QUARTERLY'
    class_id INTEGER NOT NULL REFERENCES classes(class_id) ON DELETE CASCADE,
    academic_year VARCHAR(10) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    status VARCHAR(20) DEFAULT 'SCHEDULED' CHECK (status IN ('SCHEDULED','ONGOING','COMPL
    created_at TIMESTAMP DEFAULT NOW(),
```

```
    updated_at TIMESTAMP
);

-- Exam schedule
CREATE TABLE exam_schedule (
    schedule_id SERIAL PRIMARY KEY,
    exam_id INTEGER NOT NULL REFERENCES exams(exam_id) ON DELETE CASCADE,
    subject_id INTEGER NOT NULL REFERENCES subjects(subject_id) ON DELETE CASCADE,
    exam_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    max_marks INTEGER NOT NULL,
    pass_marks INTEGER NOT NULL,
    room_number VARCHAR(20),
    invigilator_id INTEGER REFERENCES teachers(teacher_id),
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(exam_id, subject_id)
);

-- Exam results
CREATE TABLE exam_results (
    result_id SERIAL PRIMARY KEY,
    exam_id INTEGER NOT NULL REFERENCES exams(exam_id) ON DELETE CASCADE,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    subject_id INTEGER NOT NULL REFERENCES subjects(subject_id) ON DELETE CASCADE,
    marks_obtained DECIMAL(5,2) NOT NULL,
    max_marks INTEGER NOT NULL,
    grade VARCHAR(2),
    remarks TEXT,
    entered_by_teacher_id INTEGER REFERENCES teachers(teacher_id),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP,
    UNIQUE(exam_id, student_id, subject_id)
);

-- Report cards
CREATE TABLE report_cards (
    report_card_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES students(student_id) ON DELETE CASCADE,
    exam_id INTEGER NOT NULL REFERENCES exams(exam_id) ON DELETE CASCADE,
    total_marks DECIMAL(7,2) NOT NULL,
    obtained_marks DECIMAL(7,2) NOT NULL,
    percentage DECIMAL(5,2) NOT NULL,
    grade VARCHAR(2),
    rank INTEGER,
    attendance_percentage DECIMAL(5,2),
    remarks TEXT,
    class_teacher_remarks TEXT,
    principal_remarks TEXT,
    published_date DATE,
    status VARCHAR(20) DEFAULT 'DRAFT' CHECK (status IN ('DRAFT','PUBLISHED')),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP,
    UNIQUE(student_id, exam_id)
);
```

### Library Service Tables

```
-- Books
CREATE TABLE books (
    book_id SERIAL PRIMARY KEY,
    isbn VARCHAR(17) UNIQUE,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(200) NOT NULL,
    publisher VARCHAR(200),
    publication_year INTEGER,
    category VARCHAR(100),
    total_copies INTEGER DEFAULT 1,
    available_copies INTEGER DEFAULT 1,
    price DECIMAL(8,2),
    language VARCHAR(50),
    shelf_location VARCHAR(50),
    added_date DATE DEFAULT NOW(),
    status VARCHAR(20) DEFAULT 'AVAILABLE' CHECK (status IN ('AVAILABLE','OUT_OF_STOCK','
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Library issues
CREATE TABLE library_issues (
    issue_id SERIAL PRIMARY KEY,
    book_id INTEGER NOT NULL REFERENCES books(book_id) ON DELETE CASCADE,
    student_id INTEGER REFERENCES students(student_id) ON DELETE CASCADE,
    teacher_id INTEGER REFERENCES teachers(teacher_id) ON DELETE CASCADE,
    issued_date DATE NOT NULL DEFAULT NOW(),
    due_date DATE NOT NULL,
    return_date DATE,
    status VARCHAR(20) DEFAULT 'ISSUED' CHECK (status IN ('ISSUED','RETURNED','OVERDUE','
    issued_by_librarian_id INTEGER REFERENCES users(user_id),
    returned_to_librarian_id INTEGER REFERENCES users(user_id),
    remarks TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Library fines
CREATE TABLE library_fines (
    fine_id SERIAL PRIMARY KEY,
    issue_id INTEGER NOT NULL REFERENCES library_issues(issue_id) ON DELETE CASCADE,
    fine_amount DECIMAL(8,2) NOT NULL,
    fine_reason VARCHAR(100),
    days_overdue INTEGER,
    payment_status VARCHAR(20) DEFAULT 'PENDING' CHECK (payment_status IN ('PENDING','PAI
    payment_date DATE,
    payment_mode VARCHAR(20),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Book requests
CREATE TABLE book_requests (
    request_id SERIAL PRIMARY KEY,
```

```
    user_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    book_title VARCHAR(255) NOT NULL,
    author VARCHAR(200),
    isbn VARCHAR(17),
    reason TEXT,
    status VARCHAR(20) DEFAULT 'PENDING' CHECK (status IN ('PENDING','APPROVED','REJECTED
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);
```

### Principal Dashboard Service Tables

```
-- Announcements
CREATE TABLE announcements (
    announcement_id SERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    content TEXT NOT NULL,
    announcement_type VARCHAR(50) CHECK (announcement_type IN ('GENERAL','ACADEMIC','EVEN
    target_audience VARCHAR(50) CHECK (target_audience IN ('ALL','STUDENTS','TEACHERS','P
    created_by_user_id INTEGER NOT NULL REFERENCES users(user_id),
    start_date DATE NOT NULL,
    end_date DATE,
    priority VARCHAR(10) DEFAULT 'MEDIUM' CHECK (priority IN ('LOW','MEDIUM','HIGH')),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Analytics cache (for performance)
CREATE TABLE analytics_cache (
    cache_id SERIAL PRIMARY KEY,
    metric_name VARCHAR(100) NOT NULL,
    metric_value TEXT NOT NULL,
    metric_date DATE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(metric_name, metric_date)
);
```

### Support Ticket Service Tables

```
-- Support tickets
CREATE TABLE support_tickets (
    ticket_id SERIAL PRIMARY KEY,
    ticket_number VARCHAR(50) UNIQUE NOT NULL,
    reported_by_user_id INTEGER NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    category VARCHAR(50) NOT NULL CHECK (category IN ('TECHNICAL','ACADEMIC','FINANCIAL',
    priority VARCHAR(10) DEFAULT 'MEDIUM' CHECK (priority IN ('LOW','MEDIUM','HIGH','URGE
    subject VARCHAR(200) NOT NULL,
    description TEXT NOT NULL,
    status VARCHAR(20) DEFAULT 'OPEN' CHECK (status IN ('OPEN','IN_PROGRESS','RESOLVED','
    assigned_to_admin_id INTEGER REFERENCES users(user_id),
```

```
    resolution TEXT,
    resolved_date TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);

-- Ticket comments
CREATE TABLE ticket_comments (
    comment_id SERIAL PRIMARY KEY,
    ticket_id INTEGER NOT NULL REFERENCES support_tickets(ticket_id) ON DELETE CASCADE,
    user_id INTEGER NOT NULL REFERENCES users(user_id),
    comment TEXT NOT NULL,
    is_internal BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Ticket attachments
CREATE TABLE ticket_attachments (
    attachment_id SERIAL PRIMARY KEY,
    ticket_id INTEGER NOT NULL REFERENCES support_tickets(ticket_id) ON DELETE CASCADE,
    file_name VARCHAR(255) NOT NULL,
    file_url VARCHAR(500) NOT NULL,
    file_size INTEGER,
    uploaded_by_user_id INTEGER REFERENCES users(user_id),
    uploaded_at TIMESTAMP DEFAULT NOW()
);
```

### Notification Service Tables

```
-- Notifications
CREATE TABLE notifications (
    notification_id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(user_id) ON DELETE CASCADE,
    title VARCHAR(200) NOT NULL,
    message TEXT NOT NULL,
    notification_type VARCHAR(50) CHECK (notification_type IN ('INFO','WARNING','SUCCESS'
    channel VARCHAR(20) CHECK (channel IN ('IN_APP','EMAIL','SMS','PUSH')),
    read_status BOOLEAN DEFAULT FALSE,
    read_at TIMESTAMP,
    link_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT NOW()
);

-- Notification templates
CREATE TABLE notification_templates (
    template_id SERIAL PRIMARY KEY,
    template_name VARCHAR(100) UNIQUE NOT NULL,
    subject VARCHAR(200),
    body TEXT NOT NULL,
    channel VARCHAR(20) CHECK (channel IN ('EMAIL','SMS','PUSH')),
    variables TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP
);
```

```
-- Email logs
CREATE TABLE email_logs (
    log_id SERIAL PRIMARY KEY,
    to_email VARCHAR(200) NOT NULL,
    from_email VARCHAR(200),
    subject VARCHAR(200),
    body TEXT,
    status VARCHAR(20) CHECK (status IN ('SENT','FAILED','PENDING')),
    error_message TEXT,
    sent_at TIMESTAMP DEFAULT NOW()
);

-- SMS logs
CREATE TABLE sms_logs (
    log_id SERIAL PRIMARY KEY,
    to_phone VARCHAR(15) NOT NULL,
    message TEXT NOT NULL,
    status VARCHAR(20) CHECK (status IN ('SENT','FAILED','PENDING')),
    error_message TEXT,
    sent_at TIMESTAMP DEFAULT NOW()
);
```

This comprehensive document provides complete LLD diagrams, all API endpoints with their functionalities, and full PostgreSQL database schemas for every service in the School Management System microservices architecture. [1] [2] [3] [4] [5] [6] [7] [8]

⁂


1. https://www.geeksforgeeks.org/software-engineering/class-diagram-for-school-management-system/
2. https://www.slideshare.net/slideshow/student-management-system-report/54344637
3. https://www.scribd.com/document/448897589/School-Management-System-UML-Diagram-FreeProjectz
4. https://www.geeksforgeeks.org/system-design/system-design-for-library-management/
5. https://github.com/emanuel0303/school-management-api-java-spring-boot
6. https://www.codewithmurad.com/2025/06/school-management-system-project-springboot-reactjs-mysql.html
7. https://fullstackwithsangeetha.hashnode.dev/student-management-system-with-spring-boot-and-postgresql
8. https://dev.to/mohammedquasimda/building-a-student-management-api-with-spring-boot-a-step-by-step-guide-47aa