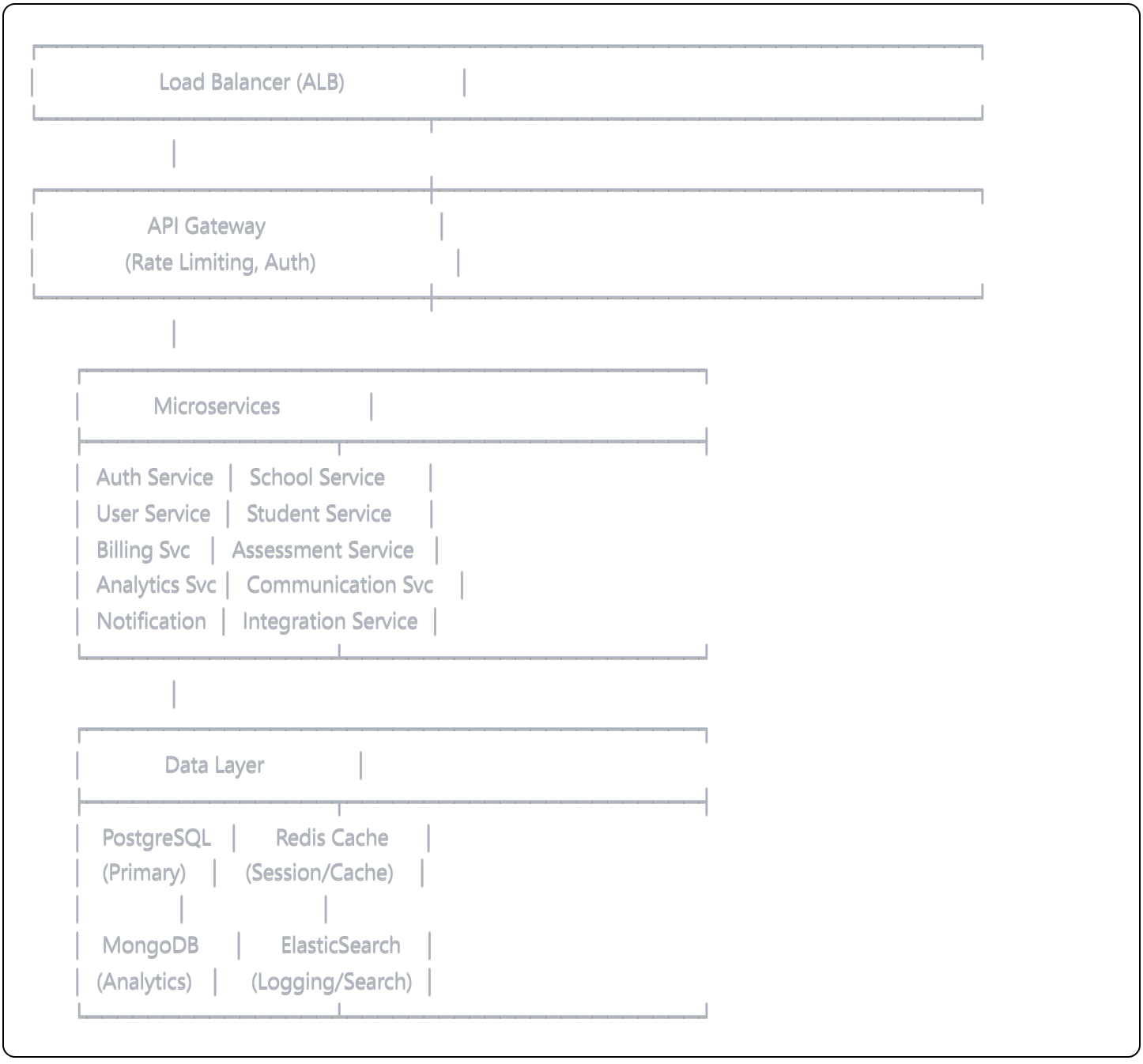
Teacher Dashboard Technical Implementation Plan

Architecture Overview

System Architecture



Technology Stack

Backend Services

- **Runtime:** Node.js 18+ with TypeScript
- **Framework:** NestJS (for microservices architecture)
- **Database:** PostgreSQL 15+ (primary), MongoDB (analytics), Redis (caching)

- **Message Queue:** AWS SQS + AWS SNS
- **Search:** Elasticsearch
- **File Storage:** AWS S3
- **CDN:** AWS CloudFront

Frontend

- **Framework:** React 18+ with TypeScript
- **State Management:** Redux Toolkit + RTK Query
- **UI Library:** Material-UI (MUI) + Custom Design System
- **Mobile:** React Native (future phase)
- **Build Tool:** Vite

Infrastructure

- **Container:** Docker + Kubernetes
- **Cloud Provider:** AWS
- **CI/CD:** GitHub Actions
- **Monitoring:** DataDog + AWS CloudWatch
- **Security:** AWS WAF + Vault for secrets

Database Design & Data Models

Core Database Schema

1. Multi-Tenancy & Organizations

```sql
```

```sql
-- Organizations (Schools/Districts)
CREATE TABLE organizations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    type ENUM('school', 'district', 'network') DEFAULT 'school',
    subdomain VARCHAR(100) UNIQUE,
    custom_domain VARCHAR(255),
    status ENUM('active', 'suspended', 'trial', 'cancelled') DEFAULT 'trial',

    -- Branding
    logo_url TEXT,
    primary_color VARCHAR(7), -- hex color
    secondary_color VARCHAR(7),

    -- Contact Information
    address_line1 VARCHAR(255),
    address_line2 VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    postal_code VARCHAR(20),
    country VARCHAR(2) DEFAULT 'US',
    phone VARCHAR(20),
    website VARCHAR(255),

    -- Settings
    timezone VARCHAR(50) DEFAULT 'America/New_York',
    date_format VARCHAR(20) DEFAULT 'MM/DD/YYYY',
    academic_year_start DATE,
    academic_year_end DATE,

    -- Metadata
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by UUID,

    -- Constraints
    CONSTRAINT chk_colors CHECK (
        primary_color ~ '^#[0-9A-Fa-f]{6}$' AND
        secondary_color ~ '^#[0-9A-Fa-f]{6}$'
    )
);

-- Organization Settings (JSON configuration)
```

```sql
CREATE TABLE organization_settings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    category VARCHAR(50) NOT NULL, -- 'grading', 'communication', 'attendance', etc.
    settings JSONB NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(organization_id, category)
);

-- Subscription Management
CREATE TABLE subscriptions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    plan_name VARCHAR(50) NOT NULL, -- 'starter', 'growth', 'professional', 'enterprise'
    status ENUM('active', 'cancelled', 'past_due', 'unpaid') DEFAULT 'active',

    -- Pricing
    base_price_cents INTEGER NOT NULL,
    per_teacher_price_cents INTEGER NOT NULL,
    billing_cycle ENUM('monthly', 'annually') DEFAULT 'monthly',

    -- Limits
    max_teachers INTEGER,
    max_students INTEGER,
    data_retention_months INTEGER,
    api_calls_per_month INTEGER,

    -- Billing
    stripe_subscription_id VARCHAR(255),
    current_period_start TIMESTAMP,
    current_period_end TIMESTAMP,
    trial_end TIMESTAMP,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Usage Tracking
CREATE TABLE usage_metrics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    metric_type VARCHAR(50) NOT NULL, -- 'api_calls', 'storage_gb', 'active_teachers'
```

```
    metric_value INTEGER NOT NULL,
    measured_at TIMESTAMP NOT NULL,
    billing_period VARCHAR(7) NOT NULL, -- 'YYYY-MM'

    INDEX idx_usage_org_period (organization_id, billing_period),
    INDEX idx_usage_measured_at (measured_at)
);
```

2. User Management & Authentication

sql

```sql
-- Users (Teachers, Admins, etc.)
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Authentication
    email VARCHAR(255) NOT NULL,
    password_hash VARCHAR(255), -- nullable for SSO users
    email_verified BOOLEAN DEFAULT FALSE,
    phone VARCHAR(20),
    phone_verified BOOLEAN DEFAULT FALSE,

    -- Profile Information
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    display_name VARCHAR(200),
    title VARCHAR(100), -- 'Teacher', 'Principal', 'Administrator'
    avatar_url TEXT,

    -- Role & Permissions
    role ENUM('super_admin', 'org_admin', 'teacher', 'substitute', 'observer') DEFAULT 'teacher',
    permissions JSONB, -- granular permissions

    -- Settings
    timezone VARCHAR(50),
    language VARCHAR(5) DEFAULT 'en-US',
    preferences JSONB,

    -- Status
    status ENUM('active', 'inactive', 'suspended', 'pending') DEFAULT 'pending',
    last_login_at TIMESTAMP,
    last_active_at TIMESTAMP,

    -- Metadata
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by UUID,

    -- Constraints
    UNIQUE(organization_id, email),
    INDEX idx_users_org_email (organization_id, email),
    INDEX idx_users_org_role (organization_id, role)
);
```

```sql
-- User Sessions
CREATE TABLE user_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    session_token VARCHAR(255) NOT NULL UNIQUE,
    refresh_token VARCHAR(255) NOT NULL UNIQUE,

    -- Session Info
    ip_address INET,
    user_agent TEXT,
    device_type VARCHAR(50),

    -- Lifecycle
    expires_at TIMESTAMP NOT NULL,
    last_accessed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_sessions_token (session_token),
    INDEX idx_sessions_user (user_id),
    INDEX idx_sessions_expires (expires_at)
);

-- User Invitations
CREATE TABLE user_invitations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    email VARCHAR(255) NOT NULL,
    role ENUM('org_admin', 'teacher', 'substitute', 'observer') DEFAULT 'teacher',
    invited_by UUID NOT NULL REFERENCES users(id),

    -- Invitation Details
    invitation_token VARCHAR(255) NOT NULL UNIQUE,
    expires_at TIMESTAMP NOT NULL,
    accepted_at TIMESTAMP,
    accepted_by UUID REFERENCES users(id),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_invitations_token (invitation_token),
    INDEX idx_invitations_org (organization_id)
);
```

3. Academic Structure

sql

```sql
-- Academic Years
CREATE TABLE academic_years (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL, -- '2024-2025', 'Fall 2024'
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    is_active BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(organization_id, name),
    INDEX idx_academic_years_org (organization_id, is_active)
);

-- Grade Levels
CREATE TABLE grade_levels (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    name VARCHAR(50) NOT NULL, -- 'Kindergarten', '1st Grade', '9th Grade'
    code VARCHAR(10) NOT NULL, -- 'K', '1', '9'
    sort_order INTEGER NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(organization_id, code),
    INDEX idx_grade_levels_org (organization_id, is_active)
);

-- Subjects
CREATE TABLE subjects (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL, -- 'Mathematics', 'English Language Arts'
    code VARCHAR(20) NOT NULL, -- 'MATH', 'ELA'
    description TEXT,
    color VARCHAR(7), -- hex color for UI
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(organization_id, code),
    INDEX idx_subjects_org (organization_id, is_active)
);
```

```sql
-- Classes/Courses
CREATE TABLE classes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    academic_year_id UUID NOT NULL REFERENCES academic_years(id) ON DELETE CASCADE,
    subject_id UUID NOT NULL REFERENCES subjects(id) ON DELETE CASCADE,
    grade_level_id UUID NOT NULL REFERENCES grade_levels(id) ON DELETE CASCADE,

    -- Basic Information
    name VARCHAR(255) NOT NULL, -- 'Advanced Algebra - Period 3'
    code VARCHAR(50), -- 'ALG2-P3'
    description TEXT,

    -- Schedule
    period VARCHAR(20),
    room_number VARCHAR(20),
    schedule JSONB, -- {'monday': '09:00-09:50', 'tuesday': '09:00-09:50', ...}

    -- Teachers (primary and co-teachers)
    primary_teacher_id UUID NOT NULL REFERENCES users(id),

    -- Settings
    max_enrollment INTEGER DEFAULT 30,
    is_active BOOLEAN DEFAULT TRUE,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_classes_org_year (organization_id, academic_year_id),
    INDEX idx_classes_teacher (primary_teacher_id)
);

-- Class Co-Teachers (many-to-many)
CREATE TABLE class_teachers (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    class_id UUID NOT NULL REFERENCES classes(id) ON DELETE CASCADE,
    teacher_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    role ENUM('primary', 'co_teacher', 'assistant', 'substitute') DEFAULT 'co_teacher',
    start_date DATE,
    end_date DATE,

    UNIQUE(class_id, teacher_id),
    INDEX idx_class_teachers_class (class_id),
```

```
    INDEX idx_class_teachers_teacher (teacher_id)
);
```

4. Student Management

sql

```sql
-- Students
CREATE TABLE students (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Personal Information
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    middle_name VARCHAR(100),
    preferred_name VARCHAR(100),
    date_of_birth DATE,
    gender VARCHAR(20),

    -- Identification
    student_id VARCHAR(50), -- school-specific ID
    state_id VARCHAR(50), -- state student ID

    -- Contact Information
    email VARCHAR(255),
    phone VARCHAR(20),
    address_line1 VARCHAR(255),
    address_line2 VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    postal_code VARCHAR(20),

    -- Academic Information
    grade_level_id UUID REFERENCES grade_levels(id),
    enrollment_date DATE,
    graduation_date DATE,

    -- Special Programs
    special_education BOOLEAN DEFAULT FALSE,
    english_language_learner BOOLEAN DEFAULT FALSE,
    gifted_and_talented BOOLEAN DEFAULT FALSE,
    free_reduced_lunch BOOLEAN DEFAULT FALSE,

    -- Status
    status ENUM('enrolled', 'transferred', 'graduated', 'dropped', 'inactive') DEFAULT 'enrolled',

    -- Metadata
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    UNIQUE(organization_id, student_id),
    INDEX idx_students_org_status (organization_id, status),
    INDEX idx_students_org_grade (organization_id, grade_level_id)
);

-- Student Class Enrollments
CREATE TABLE student_enrollments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    class_id UUID NOT NULL REFERENCES classes(id) ON DELETE CASCADE,

    enrollment_date DATE NOT NULL,
    withdrawal_date DATE,
    status ENUM('enrolled', 'withdrawn', 'transferred') DEFAULT 'enrolled',

    -- Academic tracking
    final_grade VARCHAR(5),
    credit_hours DECIMAL(3,1),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(student_id, class_id),
    INDEX idx_enrollments_student (student_id),
    INDEX idx_enrollments_class (class_id, status)
);

-- Student Guardians/Parents
CREATE TABLE student_guardians (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,

    -- Personal Information
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    relationship VARCHAR(50) NOT NULL, -- 'parent', 'guardian', 'grandparent', etc.

    -- Contact Information
    email VARCHAR(255),
    primary_phone VARCHAR(20),
    secondary_phone VARCHAR(20),
    address_line1 VARCHAR(255),
    address_line2 VARCHAR(255),
```

```
    city VARCHAR(100),
    state VARCHAR(100),
    postal_code VARCHAR(20),

    -- Preferences
    primary_contact BOOLEAN DEFAULT FALSE,
    emergency_contact BOOLEAN DEFAULT FALSE,
    pickup_authorized BOOLEAN DEFAULT TRUE,

    -- Communication Preferences
    email_notifications BOOLEAN DEFAULT TRUE,
    sms_notifications BOOLEAN DEFAULT FALSE,
    preferred_language VARCHAR(5) DEFAULT 'en-US',

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_guardians_student (student_id),
    INDEX idx_guardians_email (email)
);
```

5. Assessment & Grading System

sql

```sql
-- Grading Periods
CREATE TABLE grading_periods (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    academic_year_id UUID NOT NULL REFERENCES academic_years(id) ON DELETE CASCADE,

    name VARCHAR(100) NOT NULL, -- 'Quarter 1', 'Semester 1', 'Trimester 2'
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    weight DECIMAL(5,2) DEFAULT 25.00, -- percentage weight
    is_active BOOLEAN DEFAULT TRUE,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_grading_periods_org_year (organization_id, academic_year_id)
);

-- Assignment Categories
CREATE TABLE assignment_categories (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    class_id UUID NOT NULL REFERENCES classes(id) ON DELETE CASCADE,

    name VARCHAR(100) NOT NULL, -- 'Homework', 'Tests', 'Projects', 'Participation'
    description TEXT,
    weight DECIMAL(5,2) NOT NULL, -- percentage weight in final grade
    color VARCHAR(7),
    sort_order INTEGER DEFAULT 0,

    -- Grading settings
    drop_lowest INTEGER DEFAULT 0, -- number of lowest scores to drop

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_categories_class (class_id)
);

-- Assignments
CREATE TABLE assignments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    class_id UUID NOT NULL REFERENCES classes(id) ON DELETE CASCADE,
    category_id UUID REFERENCES assignment_categories(id),
    grading_period_id UUID REFERENCES grading_periods(id),
```

```
    -- Basic Information
    title VARCHAR(255) NOT NULL,
    description TEXT,
    instructions TEXT,

    -- Grading
    points_possible DECIMAL(8,2) NOT NULL,
    grading_scale ENUM('points', 'percentage', 'letter', 'pass_fail') DEFAULT 'points',

    -- Dates
    assigned_date DATE NOT NULL,
    due_date TIMESTAMP,
    available_from TIMESTAMP,
    available_until TIMESTAMP,

    -- Settings
    allow_late_submissions BOOLEAN DEFAULT TRUE,
    late_penalty_per_day DECIMAL(5,2) DEFAULT 0,
    extra_credit BOOLEAN DEFAULT FALSE,

    -- Status
    status ENUM('draft', 'published', 'closed') DEFAULT 'draft',

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_by UUID NOT NULL REFERENCES users(id),

    INDEX idx_assignments_class (class_id, status),
    INDEX idx_assignments_due_date (due_date)
);

-- Student Grades
CREATE TABLE student_grades (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    assignment_id UUID NOT NULL REFERENCES assignments(id) ON DELETE CASCADE,

    -- Grade Information
    points_earned DECIMAL(8,2),
    points_possible DECIMAL(8,2) NOT NULL,
    letter_grade VARCHAR(5),
    percentage DECIMAL(5,2),
```

```
    -- Submission Details
    submitted_at TIMESTAMP,
    graded_at TIMESTAMP,
    late_submission BOOLEAN DEFAULT FALSE,
    excused BOOLEAN DEFAULT FALSE,

    -- Feedback
    teacher_comments TEXT,
    private_notes TEXT, -- only visible to teachers

    -- Status
    status ENUM('not_submitted', 'submitted', 'graded', 'returned') DEFAULT 'not_submitted',

    -- Grading metadata
    graded_by UUID REFERENCES users(id),

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(student_id, assignment_id),
    INDEX idx_grades_student (student_id),
    INDEX idx_grades_assignment (assignment_id),
    INDEX idx_grades_status (status)
);

-- Grade Overrides (for manual grade adjustments)
CREATE TABLE grade_overrides (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    class_id UUID NOT NULL REFERENCES classes(id) ON DELETE CASCADE,
    grading_period_id UUID REFERENCES grading_periods(id),

    -- Override details
    override_type ENUM('assignment_category', 'grading_period', 'final_grade') NOT NULL,
    original_grade VARCHAR(10),
    override_grade VARCHAR(10) NOT NULL,
    reason TEXT NOT NULL,

    -- Metadata
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    INDEX idx_overrides_student_class (student_id, class_id)
);
```

6. Attendance Management

sql

```sql
-- Attendance Records
CREATE TABLE attendance_records (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    class_id UUID NOT NULL REFERENCES classes(id) ON DELETE CASCADE,

    -- Attendance Details
    attendance_date DATE NOT NULL,
    period VARCHAR(20), -- class period or 'full_day'
    status ENUM('present', 'absent', 'tardy', 'excused_absent', 'early_dismissal') NOT NULL,

    -- Time tracking
    arrival_time TIME,
    departure_time TIME,
    minutes_late INTEGER DEFAULT 0,

    -- Notes and reasons
    absence_reason VARCHAR(255),
    teacher_notes TEXT,
    excused_by UUID REFERENCES users(id),
    excuse_reason TEXT,

    -- Metadata
    recorded_by UUID NOT NULL REFERENCES users(id),
    recorded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_by UUID REFERENCES users(id),
    updated_at TIMESTAMP,

    UNIQUE(student_id, class_id, attendance_date, period),
    INDEX idx_attendance_student_date (student_id, attendance_date),
    INDEX idx_attendance_class_date (class_id, attendance_date),
    INDEX idx_attendance_date_status (attendance_date, status)
);

-- Attendance Patterns (for analytics)
CREATE TABLE attendance_patterns (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    academic_year_id UUID NOT NULL REFERENCES academic_years(id) ON DELETE CASCADE,

    -- Calculated metrics (updated daily)
    total_days INTEGER DEFAULT 0,
    present_days INTEGER DEFAULT 0,
```

```sql
    absent_days INTEGER DEFAULT 0,
    tardy_days INTEGER DEFAULT 0,
    excused_days INTEGER DEFAULT 0,

    -- Percentages
    attendance_rate DECIMAL(5,2) DEFAULT 0.00,
    tardiness_rate DECIMAL(5,2) DEFAULT 0.00,

    -- Streak tracking
    current_present_streak INTEGER DEFAULT 0,
    current_absent_streak INTEGER DEFAULT 0,
    longest_present_streak INTEGER DEFAULT 0,
    longest_absent_streak INTEGER DEFAULT 0,

    -- Risk indicators
    chronic_absence_risk BOOLEAN DEFAULT FALSE,
    truancy_risk BOOLEAN DEFAULT FALSE,

    last_calculated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UNIQUE(student_id, academic_year_id),
    INDEX idx_patterns_student (student_id),
    INDEX idx_patterns_risk (chronic_absence_risk, truancy_risk)
);
```

7. Behavior Management

sql

```sql
-- Behavior Categories
CREATE TABLE behavior_categories (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    name VARCHAR(100) NOT NULL, -- 'Academic', 'Social', 'Safety'
    type ENUM('positive', 'negative', 'neutral') NOT NULL,
    color VARCHAR(7),
    icon VARCHAR(50),
    points INTEGER DEFAULT 0, -- point value for behavior
    is_active BOOLEAN DEFAULT TRUE,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_behavior_categories_org (organization_id, type)
);

-- Behavior Incidents
CREATE TABLE behavior_incidents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,
    class_id UUID REFERENCES classes(id), -- nullable for school-wide incidents
    category_id UUID NOT NULL REFERENCES behavior_categories(id),

    -- Incident Details
    incident_date TIMESTAMP NOT NULL,
    location VARCHAR(255), -- classroom, hallway, cafeteria, etc.
    description TEXT NOT NULL,
    severity ENUM('minor', 'moderate', 'major', 'severe') DEFAULT 'minor',

    -- Actions Taken
    action_taken TEXT,
    follow_up_required BOOLEAN DEFAULT FALSE,
    follow_up_date DATE,
    follow_up_notes TEXT,

    -- Parent Notification
    parent_notified BOOLEAN DEFAULT FALSE,
    parent_notification_method VARCHAR(50),
    parent_notification_date TIMESTAMP,

    -- Administrative Review
    admin_reviewed BOOLEAN DEFAULT FALSE,
```

```
    admin_notes TEXT,
    admin_action TEXT,
    reviewed_by UUID REFERENCES users(id),
    reviewed_at TIMESTAMP,

    -- Metadata
    reported_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_incidents_student (student_id, incident_date),
    INDEX idx_incidents_class (class_id, incident_date),
    INDEX idx_incidents_severity (severity, incident_date)
);

-- Behavior Interventions
CREATE TABLE behavior_interventions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    student_id UUID NOT NULL REFERENCES students(id) ON DELETE CASCADE,

    -- Intervention Details
    intervention_type VARCHAR(100) NOT NULL, -- 'BIP', 'Counseling', 'Mentoring'
    start_date DATE NOT NULL,
    end_date DATE,
    status ENUM('active', 'completed', 'discontinued') DEFAULT 'active',

    -- Goals and Objectives
    goals TEXT,
    strategies TEXT,
    success_criteria TEXT,

    -- Tracking
    progress_notes TEXT,
    effectiveness_rating INTEGER CHECK (effectiveness_rating BETWEEN 1 AND 5),

    -- Responsibility
    assigned_to UUID NOT NULL REFERENCES users(id),
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_interventions_student (student_id, status),
```

```
    INDEX idx_interventions_assigned (assigned_to, status)
);
```

8. Communication System

sql

```sql
-- Communication Threads
CREATE TABLE communication_threads (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Thread Information
    subject VARCHAR(255) NOT NULL,
    thread_type ENUM('teacher_parent', 'teacher_admin', 'general', 'announcement') DEFAULT 'general',
    priority ENUM('low', 'normal', 'high', 'urgent') DEFAULT 'normal',

    -- Participants
    created_by UUID NOT NULL REFERENCES users(id),

    -- Related entities
    student_id UUID REFERENCES students(id), -- if student-related
    class_id UUID REFERENCES classes(id), -- if class-related

    -- Status
    status ENUM('active', 'closed', 'archived') DEFAULT 'active',
    is_read BOOLEAN DEFAULT FALSE,

    -- Metadata
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_message_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_threads_org (organization_id, status),
    INDEX idx_threads_student (student_id, status),
    INDEX idx_threads_class (class_id, status),
    INDEX idx_threads_updated (updated_at)
);

-- Thread Participants
CREATE TABLE thread_participants (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    thread_id UUID NOT NULL REFERENCES communication_threads(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id), -- nullable for guardian participants
    guardian_id UUID REFERENCES student_guardians(id), -- nullable for user participants

    role ENUM('owner', 'participant', 'observer') DEFAULT 'participant',
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    left_at TIMESTAMP,
```

```
    -- Read status
    last_read_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    unread_count INTEGER DEFAULT 0,

    -- Notifications
    notifications_enabled BOOLEAN DEFAULT TRUE,

    UNIQUE(thread_id, user_id),
    UNIQUE(thread_id, guardian_id),
    INDEX idx_participants_thread (thread_id),
    INDEX idx_participants_user (user_id),
    INDEX idx_participants_guardian (guardian_id)
);

-- Messages
CREATE TABLE messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    thread_id UUID NOT NULL REFERENCES communication_threads(id) ON DELETE CASCADE,

    -- Sender (either user or guardian)
    sender_user_id UUID REFERENCES users(id),
    sender_guardian_id UUID REFERENCES student_guardians(id),

    -- Message Content
    content TEXT NOT NULL,
    message_type ENUM('text', 'attachment', 'system') DEFAULT 'text',

    -- Attachments
    attachments JSONB, -- array of attachment objects

    -- Status
    is_edited BOOLEAN DEFAULT FALSE,
    edited_at TIMESTAMP,
    is_deleted BOOLEAN DEFAULT FALSE,
    deleted_at TIMESTAMP,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_messages_thread (thread_id, created_at),
    INDEX idx_messages_sender_user (sender_user_id),
    INDEX idx_messages_sender_guardian (sender_guardian_id)
);

-- Announcements
```

```sql
CREATE TABLE announcements (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Announcement Details
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    announcement_type ENUM('general', 'emergency', 'academic', 'event') DEFAULT 'general',
    priority ENUM('low', 'normal', 'high', 'urgent') DEFAULT 'normal',

    -- Targeting
    target_audience JSONB, -- roles, grades, classes to target

    -- Scheduling
    publish_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP,

    -- Status
    status ENUM('draft', 'published', 'expired', 'archived') DEFAULT 'draft',

    -- Metadata
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_announcements_org_status (organization_id, status),
    INDEX idx_announcements_publish (publish_at)
);
```

9. Analytics & Reporting

sql

```sql
-- Report Templates
CREATE TABLE report_templates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Template Information
    name VARCHAR(255) NOT NULL,
    description TEXT,
    category VARCHAR(100) NOT NULL, -- 'academic', 'attendance', 'behavior'
    report_type ENUM('student', 'class', 'grade_level', 'school') NOT NULL,

    -- Configuration
    config JSONB NOT NULL, -- report parameters and settings
    sql_template TEXT, -- parameterized SQL for custom reports

    -- Permissions
    is_public BOOLEAN DEFAULT FALSE, -- available to all users in org
    created_by UUID NOT NULL REFERENCES users(id),

    -- Usage
    usage_count INTEGER DEFAULT 0,
    last_used_at TIMESTAMP,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_templates_org_category (organization_id, category),
    INDEX idx_templates_org_public (organization_id, is_public)
);

-- Generated Reports
CREATE TABLE generated_reports (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    template_id UUID REFERENCES report_templates(id),

    -- Report Details
    name VARCHAR(255) NOT NULL,
    description TEXT,
    parameters JSONB, -- filters and parameters used

    -- Output
    file_url TEXT, -- S3 URL to generated file
```

```
    file_format VARCHAR(20), -- 'pdf', 'xlsx', 'csv'
    file_size_bytes INTEGER,

    -- Status
    status ENUM('pending', 'generating', 'completed', 'failed') DEFAULT 'pending',
    error_message TEXT,

    -- Metadata
    generated_by UUID NOT NULL REFERENCES users(id),
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP, -- auto-cleanup old reports

    INDEX idx_reports_org_status (organization_id, status),
    INDEX idx_reports_generated_by (generated_by),
    INDEX idx_reports_expires (expires_at)
);

-- Analytics Events (for tracking user behavior)
CREATE TABLE analytics_events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id),

    -- Event Details
    event_type VARCHAR(100) NOT NULL, -- 'page_view', 'button_click', 'report_generated'
    event_name VARCHAR(100) NOT NULL,
    properties JSONB, -- additional event data

    -- Context
    session_id UUID,
    page_url TEXT,
    user_agent TEXT,
    ip_address INET,

    -- Timing
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_events_org_type (organization_id, event_type),
    INDEX idx_events_user (user_id, timestamp),
    INDEX idx_events_timestamp (timestamp)
);
```

10. Integration & API Management

sql

```sql
-- API Keys
CREATE TABLE api_keys (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Key Information
    name VARCHAR(255) NOT NULL,
    key_hash VARCHAR(255) NOT NULL UNIQUE, -- hashed API key
    key_prefix VARCHAR(20) NOT NULL, -- first 8 chars for identification

    -- Permissions
    scopes JSONB NOT NULL, -- array of allowed scopes
    rate_limit_per_hour INTEGER DEFAULT 1000,

    -- Status
    is_active BOOLEAN DEFAULT TRUE,
    last_used_at TIMESTAMP,

    -- Metadata
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP,

    INDEX idx_api_keys_org (organization_id, is_active),
    INDEX idx_api_keys_prefix (key_prefix)
);

-- API Usage Logs
CREATE TABLE api_usage_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    api_key_id UUID REFERENCES api_keys(id),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Request Details
    endpoint VARCHAR(255) NOT NULL,
    method VARCHAR(10) NOT NULL,
    status_code INTEGER NOT NULL,
    response_time_ms INTEGER,

    -- Request/Response Data
    request_size_bytes INTEGER,
    response_size_bytes INTEGER,
```

```
    -- Client Information
    ip_address INET,
    user_agent TEXT,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_api_logs_key (api_key_id, created_at),
    INDEX idx_api_logs_org_date (organization_id, created_at),
    INDEX idx_api_logs_endpoint (endpoint, created_at)
);

-- Third-party Integrations
CREATE TABLE integrations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id) ON DELETE CASCADE,

    -- Integration Details
    provider VARCHAR(100) NOT NULL, -- 'google_classroom', 'canvas', 'powerschool'
    integration_type VARCHAR(50) NOT NULL, -- 'sis', 'lms', 'communication'

    -- Configuration
    config JSONB NOT NULL, -- provider-specific configuration
    credentials_encrypted TEXT, -- encrypted credentials

    -- Sync Settings
    sync_enabled BOOLEAN DEFAULT TRUE,
    sync_frequency VARCHAR(20) DEFAULT 'daily', -- 'hourly', 'daily', 'weekly'
    last_sync_at TIMESTAMP,
    next_sync_at TIMESTAMP,

    -- Status
    status ENUM('active', 'error', 'disabled') DEFAULT 'active',
    error_message TEXT,

    -- Metadata
    created_by UUID NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_integrations_org_provider (organization_id, provider),
    INDEX idx_integrations_next_sync (next_sync_at, sync_enabled)
);

-- Integration Sync Logs
```

```
CREATE TABLE integration_sync_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    integration_id UUID NOT NULL REFERENCES integrations(id) ON DELETE CASCADE,

    -- Sync Details
    sync_type VARCHAR(50) NOT NULL, -- 'students', 'classes', 'grades'
    status ENUM('started', 'completed', 'failed') NOT NULL,

    -- Results
    records_processed INTEGER DEFAULT 0,
    records_created INTEGER DEFAULT 0,
    records_updated INTEGER DEFAULT 0,
    records_failed INTEGER DEFAULT 0,

    -- Error Details
    error_message TEXT,
    error_details JSONB,

    -- Timing
    started_at TIMESTAMP NOT NULL,
    completed_at TIMESTAMP,
    duration_seconds INTEGER,

    INDEX idx_sync_logs_integration (integration_id, started_at),
    INDEX idx_sync_logs_status (status, started_at)
);
```

API Design & Microservices

Service Architecture

1. Authentication Service (auth-service)

typescript

```
// Authentication Service API Design
interface AuthService {
  // Authentication endpoints
  POST   /auth/login
  POST   /auth/logout
  POST   /auth/refresh
  POST   /auth/forgot-password
  POST   /auth/reset-password
  POST   /auth/verify-email

  // User management
  GET    /auth/me
  PUT    /auth/me
  POST   /auth/change-password

  // Organization context
  GET    /auth/organizations
  POST   /auth/switch-organization/:orgId

  // Session management
  GET    /auth/sessions
  DELETE /auth/sessions/:sessionId
}

// JWT Payload Structure
interface JWTPayload {
  sub: string; // user ID
  org: string; // organization ID
  role: string; // user role
  permissions: string[]; // granular permissions
  iat: number;
  exp: number;
}

// Authentication Middleware
export class AuthGuard {
  async validateToken(token: string): Promise<JWTPayload> {
    // Validate JWT and check against session store
  }

  async checkPermission(user: JWTPayload, permission: string): Promise<boolean> {
    // Check if user has specific permission
  }
```

```
  async enforceOrganizationBoundary(user: JWTPayload, resourceOrgId: string): Promise<boolean> {
    // Ensure user can only access their organization's data
  }
}
```

2. User Management Service (user-service)

typescript

```
// User Service API
interface UserService {
  // User CRUD
  GET    /users
  GET    /users/:userId
  POST   /users
  PUT    /users/:userId
  DELETE /users/:userId

  // User roles and permissions
  PUT    /users/:userId/role
  PUT    /users/:userId/permissions

  // User invitations
  POST   /users/invite
  GET    /users/invitations
  POST   /users/accept-invitation/:token

  // Bulk operations
  POST   /users/bulk-import
  POST   /users/bulk-update
}

// User DTO
interface CreateUserDTO {
  email: string;
  firstName: string;
  lastName: string;
  role: UserRole;
  permissions?: string[];
  classIds?: string[];
}

// User Service Implementation
@Injectable()
export class UserService {
  async createUser(orgId: string, userData: CreateUserDTO): Promise<User> {
    // Validate organization context
    // Create user with organization boundary
    // Send invitation email
    // Return user object
  }
```

```
  async findByOrganization(orgId: string, filters: UserFilters): Promise<PaginatedResult<User>> {
    // Apply organization filter
    // Apply additional filters
    // Return paginated results
  }
}
```

3. Student Information Service (student-service)

typescript

```
// Student Service API
interface StudentService {
  // Student management
  GET    /students
  GET    /students/:studentId
  POST   /students
  PUT    /students/:studentId
  DELETE /students/:studentId

  // Enrollment management
  GET    /students/:studentId/enrollments
  POST   /students/:studentId/enrollments
  PUT    /students/:studentId/enrollments/:enrollmentId
  DELETE /students/:studentId/enrollments/:enrollmentId

  // Guardian management
  GET    /students/:studentId/guardians
  POST   /students/:studentId/guardians
  PUT    /students/:studentId/guardians/:guardianId
  DELETE /students/:studentId/guardians/:guardianId

  // Bulk operations
  POST   /students/bulk-import
  PUT    /students/bulk-update
}

// Student DTO
interface CreateStudentDTO {
  firstName: string;
  lastName: string;
  studentId?: string;
  dateOfBirth?: string;
  gradeLevelId: string;
  guardians: CreateGuardianDTO[];
  enrollments?: CreateEnrollmentDTO[];
}

// Repository Pattern Implementation
@Injectable()
export class StudentRepository {
  async findByOrganization(
    orgId: string,
    filters: StudentFilters,
```

```
    pagination: PaginationOptions
  ): Promise<PaginatedResult<Student>> {
    const query = this.db
      .select()
      .from('students')
      .where('organization_id', orgId);

    // Apply filters
    if (filters.gradeLevelId) {
      query.where('grade_level_id', filters.gradeLevelId);
    }

    if (filters.status) {
      query.where('status', filters.status);
    }

    // Apply search
    if (filters.search) {
      query.where(function() {
        this.where('first_name', 'ilike', `%${filters.search}%`)
            .orWhere('last_name', 'ilike', `%${filters.search}%`)
            .orWhere('student_id', 'ilike', `%${filters.search}%`);
      });
    }

    // Apply pagination and return results
    return await this.paginate(query, pagination);
  }
}
```

4. Academic Service (academic-service)

typescript

```
// Academic Service API
interface AcademicService {
  // Class management
  GET    /classes
  GET    /classes/:classId
  POST   /classes
  PUT    /classes/:classId
  DELETE /classes/:classId

  // Assignment management
  GET    /classes/:classId/assignments
  GET    /assignments/:assignmentId
  POST   /classes/:classId/assignments
  PUT    /assignments/:assignmentId
  DELETE /assignments/:assignmentId

  // Grading
  GET    /assignments/:assignmentId/grades
  PUT    /assignments/:assignmentId/grades/:studentId
  POST   /assignments/:assignmentId/grades/bulk-update

  // Grade calculations
  GET    /classes/:classId/gradebook
  GET    /students/:studentId/transcript
  POST   /classes/:classId/calculate-final-grades
}

// Gradebook Service Implementation
@Injectable()
export class GradebookService {
  async calculateFinalGrade(
    studentId: string,
    classId: string,
    gradingPeriodId?: string
  ): Promise<FinalGradeCalculation> {
    // Get all assignments and grades for the period
    const assignments = await this.getAssignments(classId, gradingPeriodId);
    const grades = await this.getStudentGrades(studentId, assignments.map(a => a.id));

    // Get grading scale configuration
    const gradingConfig = await this.getGradingConfig(classId);

    // Calculate weighted average by category
```

```
    const categoryAverages = this.calculateCategoryAverages(assignments, grades, gradingConfig);

    // Calculate final grade
    const finalPercentage = this.calculateWeightedAverage(categoryAverages);
    const letterGrade = this.convertToLetterGrade(finalPercentage, gradingConfig.scale);

    return {
      percentage: finalPercentage,
      letterGrade,
      points: finalPercentage * gradingConfig.totalPoints / 100,
      categoryBreakdown: categoryAverages,
      lastUpdated: new Date()
    };
  }
}
```

5. Attendance Service (attendance-service)

typescript

```
// Attendance Service API
interface AttendanceService {
  // Daily attendance
  GET    /attendance/:date
  GET    /attendance/class/:classId/:date
  POST   /attendance/record
  PUT    /attendance/:recordId

  // Bulk operations
  POST   /attendance/bulk-record
  POST   /attendance/import-csv

  // Reports and analytics
  GET    /attendance/reports/summary
  GET    /attendance/reports/student/:studentId
  GET    /attendance/reports/chronic-absence

  // Patterns and alerts
  GET    /attendance/patterns/risk-students
  POST   /attendance/alerts/configure
}

// Attendance Analytics Implementation
@Injectable()
export class AttendanceAnalyticsService {
  async calculateAttendancePatterns(studentId: string, academicYearId: string): Promise<AttendancePattern> {
    const records = await this.attendanceRepository.findByStudentAndYear(studentId, academicYearId);

    const totalDays = records.length;
    const presentDays = records.filter(r => r.status === 'present').length;
    const absentDays = records.filter(r => r.status === 'absent' || r.status === 'excused_absent').length;
    const tardyDays = records.filter(r => r.status === 'tardy').length;

    const attendanceRate = totalDays > 0 ? (presentDays / totalDays) * 100 : 0;
    const tardiness_rate = totalDays > 0 ? (tardyDays / totalDays) * 100 : 0;

    // Calculate streaks
    const presentStreak = this.calculateCurrentStreak(records, 'present');
    const absentStreak = this.calculateCurrentStreak(records, ['absent', 'excused_absent']);

    // Risk assessment
    const chronicAbsenceRisk = attendanceRate < 90; // Below 90% attendance
    const truancyRisk = absentStreak >= 5; // 5+ consecutive absences
```

```
    return {
      totalDays,
      presentDays,
      absentDays,
      tardyDays,
      attendanceRate,
      tardiness_rate,
      currentPresentStreak: presentStreak,
      currentAbsentStreak: absentStreak,
      chronicAbsenceRisk,
      truancyRisk,
      lastCalculatedAt: new Date()
    };
  }
}
```

Database Optimization & Indexing Strategy

```sql
```

```sql
-- Performance Indexes
CREATE INDEX CONCURRENTLY idx_students_org_status_grade
ON students (organization_id, status, grade_level_id)
WHERE status = 'enrolled';

CREATE INDEX CONCURRENTLY idx_attendance_student_date_desc
ON attendance_records (student_id, attendance_date DESC);

CREATE INDEX CONCURRENTLY idx_grades_assignment_status
ON student_grades (assignment_id, status)
WHERE status IN ('submitted', 'graded');

CREATE INDEX CONCURRENTLY idx_communications_org_updated
ON communication_threads (organization_id, updated_at DESC, status);

-- Partial indexes for common queries
CREATE INDEX CONCURRENTLY idx_users_org_active
ON users (organization_id, role, last_active_at)
WHERE status = 'active';

CREATE INDEX CONCURRENTLY idx_classes_org_year_active
ON classes (organization_id, academic_year_id, primary_teacher_id)
WHERE is_active = true;

-- Composite indexes for reporting queries
CREATE INDEX CONCURRENTLY idx_behavior_incidents_reporting
ON behavior_incidents (organization_id, incident_date DESC, severity, category_id);

CREATE INDEX CONCURRENTLY idx_api_usage_billing
ON api_usage_logs (organization_id, date_trunc('month', created_at), endpoint);
```

Caching Strategy

typescript

```
// Redis Cache Configuration
interface CacheConfig {
  // User sessions and authentication
  sessions: {
    ttl: 24 * 60 * 60; // 24 hours
    prefix: 'session:';
  };

  // Organization data
  organizations: {
    ttl: 60 * 60; // 1 hour
    prefix: 'org:';
  };

  // Frequently accessed student/class data
  students: {
    ttl: 30 * 60; // 30 minutes
    prefix: 'student:';
  };

  // Grade calculations
  gradebook: {
    ttl: 10 * 60; // 10 minutes
    prefix: 'grades:';
  };

  // Report results
  reports: {
    ttl: 60 * 60; // 1 hour
    prefix: 'report:';
  };
}

// Cache Service Implementation
@Injectable()
export class CacheService {
  constructor(private redis: Redis) {}

  async get<T>(key: string, ttl?: number): Promise<T | null> {
    const cached = await this.redis.get(key);
    if (cached) {
      return JSON.parse(cached);
    }
```

```
    return null;
  }

  async set<T>(key: string, value: T, ttl: number): Promise<void> {
    await this.redis.setex(key, ttl, JSON.stringify(value));
  }

  async invalidatePattern(pattern: string): Promise<void> {
    const keys = await this.redis.keys(pattern);
    if (keys.length > 0) {
      await this.redis.del(...keys);
    }
  }

  // Organization-specific cache invalidation
  async invalidateOrganizationCache(orgId: string): Promise<void> {
    await Promise.all([
      this.invalidatePattern(`org:${orgId}:*`),
      this.invalidatePattern(`student:${orgId}:*`),
      this.invalidatePattern(`grades:${orgId}:*`),
    ]);
  }
}
```

Message Queue Architecture

typescript

```typescript
// Queue Configuration
interface QueueConfig {
  // Email notifications
  emailQueue: {
    name: 'email-notifications';
    concurrency: 5;
    attempts: 3;
    backoff: 'exponential';
  };

  // Report generation
  reportQueue: {
    name: 'report-generation';
    concurrency: 2;
    attempts: 2;
    timeout: 300000; // 5 minutes
  };

  // Data sync
  syncQueue: {
    name: 'data-sync';
    concurrency: 3;
    attempts: 5;
    backoff: 'exponential';
  };

  // Analytics processing
  analyticsQueue: {
    name: 'analytics-processing';
    concurrency: 10;
    attempts: 2;
  };
}

// Queue Processor Implementation
@Processor('email-notifications')
export class EmailQueueProcessor {
  @Process('send-parent-notification')
  async sendParentNotification(job: Job<ParentNotificationData>) {
    const { studentId, message, type } = job.data;

    try {
      // Get student and guardian information
```

```
      const student = await this.studentService.findById(studentId);
      const guardians = await this.studentService.getGuardians(studentId);

      // Send notifications to all guardians
      for (const guardian of guardians) {
        if (guardian.emailNotifications) {
          await this.emailService.sendNotification({
            to: guardian.email,
            studentName: `${student.firstName} ${student.lastName}`,
            message,
            type
          });
        }

        if (guardian.smsNotifications && guardian.primaryPhone) {
          await this.smsService.sendNotification({
            to: guardian.primaryPhone,
            message: message.substring(0, 160) // SMS length limit
          });
        }
      }

      // Update notification log
      await this.notificationService.logNotification({
        studentId,
        type,
        recipientCount: guardians.length,
        status: 'sent'
      });

    } catch (error) {
      // Log error and potentially retry
      throw new Error(`Failed to send parent notification: ${error.message}`);
    }
  }
}

// Analytics Queue Processor
@Processor('analytics-processing')
export class AnalyticsQueueProcessor {
  @Process('calculate-attendance-patterns')
  async calculateAttendancePatterns(job: Job<{ studentId: string; academicYearId: string }>) {
    const { studentId, academicYearId } = job.data;
```

```
    const patterns = await this.attendanceAnalyticsService.calculateAttendancePatterns(
      studentId,
      academicYearId
    );

    // Store calculated patterns
    await this.attendanceRepository.updatePatterns(studentId, academicYearId, patterns);

    // Trigger risk alerts if needed
    if (patterns.chronicAbsenceRisk || patterns.truancyRisk) {
      await this.queueService.add('risk-alert', {
        studentId,
        riskType: patterns.chronicAbsenceRisk ? 'chronic-absence' : 'truancy',
        severity: patterns.truancyRisk ? 'high' : 'medium'
      });
    }
  }
}
```

Security Implementation

typescript

```typescript
// Security Configuration
interface SecurityConfig {
  jwt: {
    secret: string;
    expiresIn: string;
    refreshExpiresIn: string;
  };

  encryption: {
    algorithm: 'aes-256-gcm';
    keyLength: 32;
  };

  rateLimit: {
    windowMs: 15 * 60 * 1000; // 15 minutes
    max: 100; // requests per windowMs
    skipSuccessfulRequests: false;
  };

  cors: {
    origin: string[];
    credentials: true;
    optionsSuccessStatus: 200;
  };
}

// Data Encryption Service
@Injectable()
export class EncryptionService {
  private readonly algorithm = 'aes-256-gcm';
  private readonly key: Buffer;

  constructor(private config: ConfigService) {
    this.key = crypto.scryptSync(config.get('ENCRYPTION_KEY'), 'salt', 32);
  }

  encrypt(text: string): string {
    const iv = crypto.randomBytes(16);
    const cipher = crypto.createCipher(this.algorithm, this.key, iv);

    let encrypted = cipher.update(text, 'utf8', 'hex');
    encrypted += cipher.final('hex');
```

```
    const authTag = cipher.getAuthTag();

    return `${iv.toString('hex')}:${authTag.toString('hex')}:${encrypted}`;
  }

  decrypt(encryptedData: string): string {
    const [ivHex, authTagHex, encrypted] = encryptedData.split(':');

    const iv = Buffer.from(ivHex, 'hex');
    const authTag = Buffer.from(authTagHex, 'hex');

    const decipher = crypto.createDecipher(this.algorithm, this.key, iv);
    decipher.setAuthTag(authTag);

    let decrypted = decipher.update(encrypted, 'hex', 'utf8');
    decrypted += decipher.final('utf8');

    return decrypted;
  }
}

// Audit Logging
@Injectable()
export class AuditService {
  async logAction(action: AuditAction): Promise<void> {
    const auditLog = {
      id: uuid(),
      organizationId: action.organizationId,
      userId: action.userId,
      action: action.action,
      resource: action.resource,
      resourceId: action.resourceId,
      changes: action.changes,
      ipAddress: action.ipAddress,
      userAgent: action.userAgent,
      timestamp: new Date(),
    };

    // Store in audit log table
    await this.auditRepository.create(auditLog);

    // Send to external audit system if configured
    if (this.config.get('AUDIT_EXTERNAL_ENABLED')) {
      await this.externalAuditService.send(auditLog);
```

```
    }
  }
}

// Organization Boundary Enforcement
@Injectable()
export class OrganizationGuard implements CanActivate {
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const user = request.user; // From JWT

    // Extract organization ID from request
    const resourceOrgId = this.extractOrganizationId(request);

    // Ensure user can only access their organization's data
    if (user.org !== resourceOrgId) {
      throw new ForbiddenException('Access denied: organization boundary violation');
    }

    return true;
  }

  private extractOrganizationId(request: any): string {
    // Try various sources for organization ID
    return request.params.orgId ||
           request.body.organizationId ||
           request.query.organizationId ||
           request.headers['x-organization-id'];
  }
}
```

This comprehensive technical implementation plan provides:

1. **Complete database schema** with all necessary tables and relationships
2. **Microservices architecture** with clear API boundaries
3. **Security implementation** with encryption, authentication, and authorization
4. **Performance optimization** through caching and indexing strategies
5. **Scalability features** including message queues and async processing
6. **Data isolation** ensuring multi-tenant security
7. **Integration capabilities** for third-party systems

The architecture supports the business requirements while maintaining security, performance, and scalability for a SaaS platform serving thousands of schools. '