School Management SaaS Platform

High-Level Design (HLD) Document

Version: 1.0
Date: December 2, 2025
Author: Chief Software Architect
Classification: Architecture & Design

Executive Summary

This document presents the comprehensive High-Level Design (HLD) for a **multi-tenant School Management SaaS Platform** that serves as a white-labeled solution for schools worldwide. The platform adopts a **Modular Monolith architecture** deployed via **Docker Swarm** with a **hybrid multi-tenancy strategy** combining shared resources for cost-efficiency (Basic/Premium plans) and isolated resources for high-security requirements (Custom plans).

Key Strategic Decisions:

- **Architecture:** Spring Boot Modulith (not microservices) for controlled complexity
- **Multi-Tenancy:** Hybrid approach—Shared Database (Separate Schemas) for Standard plans, Database-per-Tenant for Custom plans
- **Frontend Delivery:** Hybrid (Managed + Headless/API-First modes)
- **CMS Layer:** Directus for decoupled content management
- **Infrastructure:** Docker Swarm + Traefik Proxy for dynamic tenant routing
- **Cost Model:** FOSS stack (PostgreSQL, Valkey, Prometheus, Grafana) to maximize margins

---

1. Architectural Pattern & Strategy

1.1 Architecture Selection: Spring Boot Modulith

Why Modular Monolith, Not Microservices?
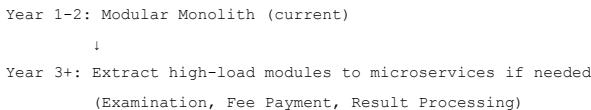
Microservices Rejected Because:

- **Operational Overhead:** Multi-tenant microservices create exponential complexity (e.g., 7 services × N tenants = 7N failure domains)
- **Data Consistency Challenges:** Distributed transactions across tenant boundaries introduce eventual consistency complexity unsuitable for academic records
- **Cost Per Tenant:** Each microservice has deployment, monitoring, and inter-service communication overhead
- **Early-Stage Agility Loss:** For a SaaS startup, microservice orchestration (Kubernetes, service mesh) diverts resources from product differentiation

Modular Monolith Selected Because:

- **Clear Module Boundaries:** Spring Modulith enforces module isolation at compile time using `@Modulith` and `@ModuleExposed` annotations
- **Unified Deployment:** Single WAR/JAR deployment simplifies tenant rollouts and infrastructure
- **Scalability:** Horizontal scaling via Docker Swarm replicas addresses concurrency (e.g., exam result spikes) without architectural fragmentation
- **Transactional Safety:** All-in-one database connection pool ensures ACID properties for critical operations (grades, fees, admissions)
- **Migration Path:** If single modules (e.g., Examination) need independent scaling later, Spring Modulith's boundaries make extraction to microservices straightforward

Evolution Path:

```
Year 1-2: Modular Monolith (current)
          ↓
Year 3+: Extract high-load modules to microservices if needed
         (Examination, Fee Payment, Result Processing)
```

1.2 Multi-Tenancy Strategy: Hybrid (Shared Schema + Database-per-Tenant)

Pattern Selection Rationale

Aspect	Shared Database + Separate Schemas	Database-per-Tenant	Our Hybrid Choice
Data Isolation	Medium (schema-level separation)	High (physical DB isolation)	High (conditional isolation)
Cost Per Tenant	Low (~$5/month)	High (~$50/month)	Balanced ($10-50/month)
Customization	Medium (schema tuning possible)	High (schema/version flexibility)	High (plan-dependent)

Aspect	Shared Database + Separate Schemas	Database-per-Tenant	Our Hybrid Choice
Compliance (FERPA/GDPR)	Moderate (single breach affects schema)	Excellent (isolated breach)	Excellent (on-demand isolation)
Operational Complexity	Medium	High (multi-DB backup/failover)	Medium (dual strategy)

Implementation Model

Tier 1: Basic & Premium Plans → Shared Database (PostgreSQL with Separate Schemas)

```
-- Logical separation within single PostgreSQL instance
CREATE SCHEMA school_001;
CREATE SCHEMA school_002;
-- Each school's data lives in its schema
-- Row-level security via tenant context
```

Benefits:

- ☐ 40-60% cost savings vs database-per-tenant
- ☐ Simplified backup/restore (single PostgreSQL instance)
- ☐ Suitable for small-to-medium schools (<5K students)
- ☐ Automatic schema versioning for feature rollouts

Tier 2: Custom Plans → Database-per-Tenant (Dedicated PostgreSQL Instance)

```
-- Physical database isolation for enterprise schools
-- school-001-prod-db.internal
-- school-002-prod-db.internal
```

Benefits:

- ☐ Maximum data isolation (regulatory compliance)
- ☐ Custom database tuning per school
- ☐ Dedicated backup SLA
- ☐ Suitable for large schools (50K+ students) or regulated institutions

Critical Security Pattern: Tenant Context Propagation
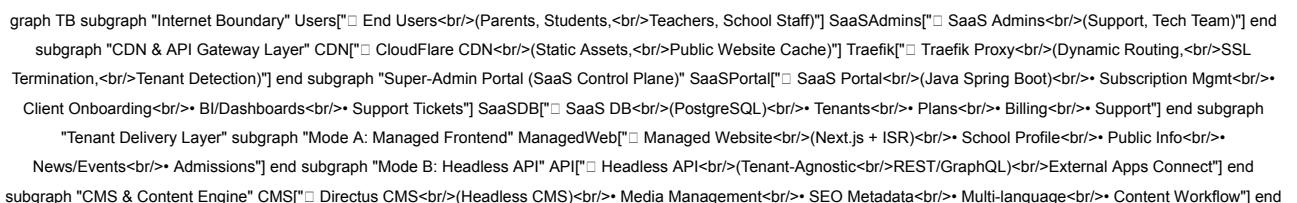
Every database query must include a tenant filter. Use **Row-Level Security (RLS)** in PostgreSQL:

```
-- Enable RLS on all tenant-aware tables
ALTER TABLE students ENABLE ROW LEVEL SECURITY;

-- Create policy: users can only see their tenant's data
CREATE POLICY student_isolation ON students
  USING (school_id = current_setting('app.current_school_id')::uuid);

-- In application layer (Spring Security):
// Set tenant context before every query
SecurityContextHolder.getContext().setAuthentication(auth);
// Extract school_id from JWT token
String schoolId = ((JwtAuthenticationToken) auth).getToken().getClaimAsString("school_id");
// Set PostgreSQL context
em.createNativeQuery("SET app.current_school_id = ?1")
  .setParameter(1, schoolId)
  .executeUpdate();
```

2. High-Level System Diagram (Mermaid.js)

graph TB subgraph "Internet Boundary" Users["☐ End Users
(Parents, Students,
Teachers, School Staff)"] SaaSAdmins["☐ SaaS Admins
(Support, Tech Team)"] end subgraph "CDN & API Gateway Layer" CDN["☐ CloudFlare CDN
(Static Assets,
Public Website Cache)"] Traefik["☐ Traefik Proxy
(Dynamic Routing,
SSL Termination,
Tenant Detection)"] end subgraph "Super-Admin Portal (SaaS Control Plane)" SaaSPortal["☐ SaaS Portal
(Java Spring Boot)
• Subscription Mgmt
• Client Onboarding
• BI/Dashboards
• Support Tickets"] SaaSDB["☐ SaaS DB
(PostgreSQL)
• Tenants
• Plans
• Billing
• Support"] end subgraph "Tenant Delivery Layer" subgraph "Mode A: Managed Frontend" ManagedWeb["☐ Managed Website
(Next.js + ISR)
• School Profile
• Public Info
• News/Events
• Admissions"] end subgraph "Mode B: Headless API" API["☐ Headless API
(Tenant-Agnostic
REST/GraphQL)
External Apps Connect"] end subgraph "CMS & Content Engine" CMS["☐ Directus CMS
(Headless CMS)
• Media Management
• SEO Metadata
• Multi-language
• Content Workflow"] end

end subgraph "Core Backend (Spring Boot Modulith)" subgraph "Module: StudentParent" StudentModule["□ Student/Parent Portal
• Dashboards
• Grades & Transcripts
• Fees & Payments
• Assignments"] end subgraph "Module: Teacher" TeacherModule["□□ Teacher Portal
• Class Mgmt
• Attendance
• Grading
• Lesson Plans"] end subgraph "Module: Examination" ExamModule["□ Exam Module
• Scheduling
• Conflict Resolution
• Question Bank
• Result Processing"] end subgraph "Module: Administration" AdminModule["⚙ Admin Module
• RBAC
• Finance/Budget
• HR/Payroll
• Student Lifecycle"] end subgraph "Module: Leadership" LeaderModule["□ Leadership BI
• Analytics Dashboard
• Enrollment Forecast
• Compliance Tracking"] end subgraph "Module: TenantConfig" TenantModule["□ Tenant Config
• Subscription Settings
• Branding/CMS
• Feature Flags
• Support Portal"] end EventBus["□ Event Bus
(Apache Kafka)
• Async Processing
• Cross-Module Events"] end subgraph "Data Layer" DB_Shared["□ PostgreSQL Shared
(Separate Schemas)
Basic/Premium Plans
school_001, school_002..."] DB_Custom["□ PostgreSQL Custom
(Database-per-Tenant)
Enterprise Plans
Dedicated Instances"] Cache["⚡ Valkey Cache
(Redis Fork)
• Session Store
• Query Cache
• Real-time Data"] FileStorage["□ MinIO/S3
(Object Storage)
• Student Photos
• Documents
• Certificates"] end subgraph "Integration Layer" PaymentGW["□ Payment Gateway
(Stripe/Razorpay)
Fee Submissions"] EmailService["□ Email/SMS Service
(SendGrid/Twilio)"] Analytics["□ Google Analytics 4
Track User Behavior"] end subgraph "Observability Stack (FOSS)" Prometheus["□ Prometheus
(Metrics Collection)"] Grafana["□ Grafana
(Dashboards)"] Loki["□ Loki
(Log Aggregation)"] Jaeger["□ Jaeger
(Distributed Tracing)"] end subgraph "Infrastructure (Docker Swarm)" Swarm["□ Docker Swarm Cluster
• Node 1 (Manager)
• Node 2 (Worker)
• Node 3 (Worker)
Auto-Scaling Replicas"] end %% Connections Users -->|Visits School Domain| CDN Users -->|API Requests| Traefik SaaSAdmins -->|Manages| SaaSPortal CDN -->|Routes Static| ManagedWeb Traefik -->|Routes by Tenant ID| StudentModule Traefik -->|Routes by Tenant ID| TeacherModule Traefik -->|Routes by Tenant ID| ExamModule Traefik -->|Routes by Tenant ID| API SaaSPortal -->|Reads Config| SaaSDB CMS -->|Serves Content| ManagedWeb CMS -->|Provides API| API StudentModule --> EventBus TeacherModule --> EventBus ExamModule --> EventBus AdminModule --> EventBus LeaderModule --> EventBus EventBus -->|Events| StudentModule StudentModule -->|Queries| Cache ExamModule -->|High-Load Queries| Cache StudentModule -->|Reads/Writes| DB_Shared TeacherModule -->|Reads/Writes| DB_Shared AdminModule -->|Reads/Writes| DB_Shared ExamModule -->|Reads/Writes| DB_Custom StudentModule -->|Stores| FileStorage AdminModule -->|Stores| FileStorage StudentModule -->|Payment| PaymentGW StudentModule -->|Notifications| EmailService ManagedWeb -->|Analytics| Analytics Prometheus -->|Scrapes Metrics| StudentModule Prometheus -->|Scrapes Metrics| ExamModule Grafana -->|Visualizes| Prometheus Loki -->|Collects Logs| StudentModule Jaeger -->|Traces Calls| EventBus StudentModule -->|Runs in| Swarm TeacherModule -->|Runs in| Swarm ExamModule -->|Runs in| Swarm Traefik -->|Routes within| Swarm style Users fill:#e1f5ff style SaaSAdmins fill:#fff3e0 style Traefik fill:#f3e5f5 style EventBus fill:#fce4ec style Cache fill:#fff9c4 style Prometheus fill:#c8e6c9 style Swarm fill:#e0e0e0

3. Technology Stack Recommendations

3.1 Frontend Layer

Component	Technology	Rationale
Managed Public Website	Next.js 14 + App Router	ISR (Incremental Static Regeneration) for SEO; supports multi-language; edge deployment via Vercel or Cloudflare
Student/Parent/Teacher Portals	React 18 + TypeScript + Vite	Fast HMR for developer experience; tree-shaking reduces bundle size
Component Library	shadcn/ui + Tailwind CSS	Accessible, customizable components; supports dark mode for accessibility compliance (WCAG 2.1 AA)
State Management	TanStack Query (React Query)	Simplifies server state management; automatic cache invalidation post-mutation
Form Validation	React Hook Form + Zod	Minimal re-renders; type-safe schema validation
Real-time Updates	Socket.io Client	Live grade notifications, attendance updates, exam result broadcasts

Dynamic Branding Implementation:

```
// Load school-specific branding from TenantConfig module
const schoolConfig = useTenantConfig(); // Hook
const theme = {
  primary: schoolConfig.brandColor,
  logo: schoolConfig.logoUrl,
  favicon: schoolConfig.faviconUrl,
  googleFont: schoolConfig.customFont
};
// Apply via Tailwind CSS custom properties
```

3.2 Backend Layer

Component	Technology	Rationale
Framework	Spring Boot 3.2 + Spring Modulith	Enterprise-grade stability; @Modulith enforces module boundaries; native GraalVM compilation for faster startup
Language	Java 21	LTS release; record types for immutability; pattern matching for cleaner code; virtual threads (Project Loom) for high concurrency
Web Framework	Spring WebFlux	Reactive, non-blocking I/O; handles exam result submissions (traffic spike) with minimal threads
Async Messaging	Apache Kafka + Spring Cloud Stream	Decouples modules; ensures result processing doesn't block API responses
API Gateway	Traefik Proxy (FOSS)	Dynamic tenant routing via Docker labels; native SSL/TLS with Let's Encrypt automation
CMS Integration	Directus SDK (Java)	Programmatic access to CMS content; supports headless queries for public website rendering

Component	Technology	Rationale
Database Access	Spring Data JPA + QueryDSL	Type-safe queries; supports custom @Query for complex reports

Module Structure (Spring Modulith):

```
// Core modules enforcing tenant isolation
@Modulith
public class StudentParentModule { }

@Modulith
public class TeacherModule { }

@Modulith
public class ExaminationModule { }

// Base module with cross-cutting concerns
@Modulith
public class TenantContextModule {
    // Tenant ID extraction, request interceptor, RLS setup
}

// Module communication via Spring's event mechanism
@Component
public class ResultPublishedListener {
    @EventListener
    public void onResultPublished(ResultPublishedEvent event) {
        // Async processing: generate certificates, send emails
    }
}
```

3.3 Database Layer

Component	Technology	Rationale
Primary Relational DB	PostgreSQL 16.x	ACID compliance for academic records; Row-Level Security (RLS) for tenant isolation; JSON/UUID native support
Cache Layer	Valkey (Redis Fork)	True open-source (Linux Foundation); drop-in Redis replacement; license-safe for commercial use
Connection Pooling	HikariCP (Spring Data default)	40% faster than other pools; supports failover configs
Search Index	Elasticsearch 8.x OR PostgreSQL Full-Text	ES for large deployments; PG FTS for cost-conscious Basic plans
Time-Series (Analytics)	PostgreSQL TimescaleDB extension	Native to PostgreSQL; stores attendance, engagement metrics with automatic compression

Database Schema Pattern:

```
-- Shared schema approach (Basic/Premium)
CREATE SCHEMA school_001;


-- Every table includes school_id + RLS
CREATE TABLE school_001.students (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL,
    name VARCHAR(255),
    email VARCHAR(255),
    enrollment_date DATE,
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES saas_db.schools(id)
);


-- RLS Policy: users can only query their school's data
ALTER TABLE school_001.students ENABLE ROW LEVEL SECURITY;
CREATE POLICY students_isolation ON school_001.students
  USING (school_id = current_setting('app.current_school_id')::uuid);


-- For Custom plans: separate database with no schema prefix
-- CREATE DATABASE school_enterprise_prod; -- dedicated PostgreSQL instance
```

3.4 Infrastructure & Deployment

Component	Technology	Rationale
Container Orchestration	Docker Swarm	Built-in Docker; no learning curve vs Kubernetes; sufficient for <1000 tenants; cost-efficient
Reverse Proxy	Traefik 3.x	Auto-discovers Docker services; dynamic SSL via Let's Encrypt; native load balancing
Container Registry	Harbor (Self-Hosted) OR GitHub Container Registry	Air-gapped deployments supported; vulnerability scanning
Persistent Storage	NFS for Swarm volumes OR MinIO (S3-compatible)	MinIO for object storage (student photos, certificates); NFS for PostgreSQL data
Monitoring	Prometheus + Grafana + Loki + Jaeger	FOSS stack; Prometheus scrapes Spring Boot actuator; Loki centralizes logs; Jaeger traces async calls
CI/CD	GitHub Actions + ArgoCD	GitOps workflow; declarative deployments; supports canary releases per tenant
Secrets Management	HashiCorp Vault OR Sealed Secrets (Kubernetes style)	Vault for centralized key management; rotates database passwords automatically

Docker Swarm Deployment:

```yaml
# docker-compose.yml (deployed via docker service create)
version: '3.9'
services:
  school-backend:
    image: schoolsaas/backend:latest
    deploy:
      replicas: 3  # Auto-scale during exams
      update_config:
        parallelism: 1
        delay: 10s
      labels:
        - "traefik.http.routers.backend.rule=Host(`api.school-platform.com`) || HostRegexp(`{subdomain:[a-z]+}.api.school-platform.co
        - "traefik.http.services.backend.loadbalancer.server.port=8080"
        - "traefik.http.services.backend.loadbalancer.healthcheck.path=/actuator/health"
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/schooldb
      - SPRING_JPA_HIBERNATE_DDL_AUTO=validate
      - SPRING_DATA_REDIS_HOST=valkey
    depends_on:
      - postgres
      - valkey

  postgres:
    image: postgres:16-alpine
    environment:
      - POSTGRES_PASSWORD_FILE=/run/secrets/db_password
    volumes:
      - db_data:/var/lib/postgresql/data
    secrets:
      - db_password

  valkey:
    image: valkey/valkey:latest
    command: valkey-server --appendonly yes

  traefik:
    image: traefik:v3.0
    command:
      - "--api.insecure=false"
      - "--providers.docker.swarmmode=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--certificatesresolvers.letsencrypt.acme.httpchallenge=true"
      - "--certificatesresolvers.letsencrypt.acme.storage=/letsencrypt/acme.json"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - letsencrypt:/letsencrypt

volumes:
  db_data:
  letsencrypt:

secrets:
  db_password:
    external: true
```

Cost Breakdown (Annual, per tenant):

Plan	Infrastructure	Monthly Cost	Annual
Basic	Shared PostgreSQL schema + Valkey + 1 replica	$5	$60
Premium	Shared PostgreSQL schema + Valkey + 3 replicas	$15	$180

Plan	Infrastructure	Monthly Cost	Annual
Custom	Dedicated PostgreSQL + backup + 5 replicas	$50	$600

4. Detailed Component Design

4.1 API Design (REST + GraphQL)

Endpoint Structure

Base URL Pattern:

```
https://api.school-platform.com/api/v1/{tenant-subdomain}/{module}/{resource}
```

Examples:

```
GET /api/v1/abc-school/students/courses  → Fetch courses for ABC School
POST /api/v1/xyz-school/exams/schedule  → Create exam for XYZ School
```

Tenant Detection via Traefik

Traefik extracts tenant ID from subdomain and injects into request headers:

```
Request: GET https://abc-school.api.school-platform.com/api/v1/students
         ↓ (Traefik middleware)
Adds Header: X-Tenant-ID: school_001
         ↓
Backend receives & sets context
```

Rate Limiting Strategy

```java
// Per-tenant rate limiting (Basic plan: 1000 req/hour, Premium: 5000)
@Component
public class TenantAwareRateLimiter {
    @Bean
    public RateLimitingPolicy rateLimitPolicy(@Value("${plan.limits}") String planJson) {
        return switch (tenantPlan) {
            case BASIC -> new RateLimitingPolicy(1000, Duration.ofHours(1));
            case PREMIUM -> new RateLimitingPolicy(5000, Duration.ofHours(1));
            case CUSTOM -> new RateLimitingPolicy(Integer.MAX_VALUE, Duration.ofHours(1));
        };
    }
}
```

API Versioning

```
/api/v1  → Current version (stable)
/api/v2  → Beta features (opt-in via feature flags)
/api/next → Experimental endpoints (internal use)
```

Mode B: Headless API for External Apps

```
// External school app connects via API key
const headers = {
  'Authorization': 'Bearer sk_live_abc123xyz...',
  'X-School-ID': 'school-uuid-here',
  'Content-Type': 'application/json'
};

// Fetch student list (external school's own data only)
const response = await fetch(
  'https://api.school-platform.com/api/v1/students?filter[status]=ACTIVE',
  { headers }
);

const students = response.json(); // ~500 results
```

API Key Security:

```
@Component
public class ApiKeyValidator {
    @Bean
    public SecurityFilterChain apiKeyChain(HttpSecurity http) {
        http.authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/v1/external/**").authenticated()
            .anyRequest().permitAll()
        ).addFilterBefore(
            new ApiKeyAuthenticationFilter(apiKeyService),
            UsernamePasswordAuthenticationFilter.class
        );
        return http.build();
    }
}

// API Key tied to school + permissions
class ApiKey {
    UUID schoolId;
    Set<Permission> permissions; // READ_STUDENTS, WRITE_GRADES, etc.
    LocalDateTime expiresAt;
    InetAddress allowedIp;
}
```
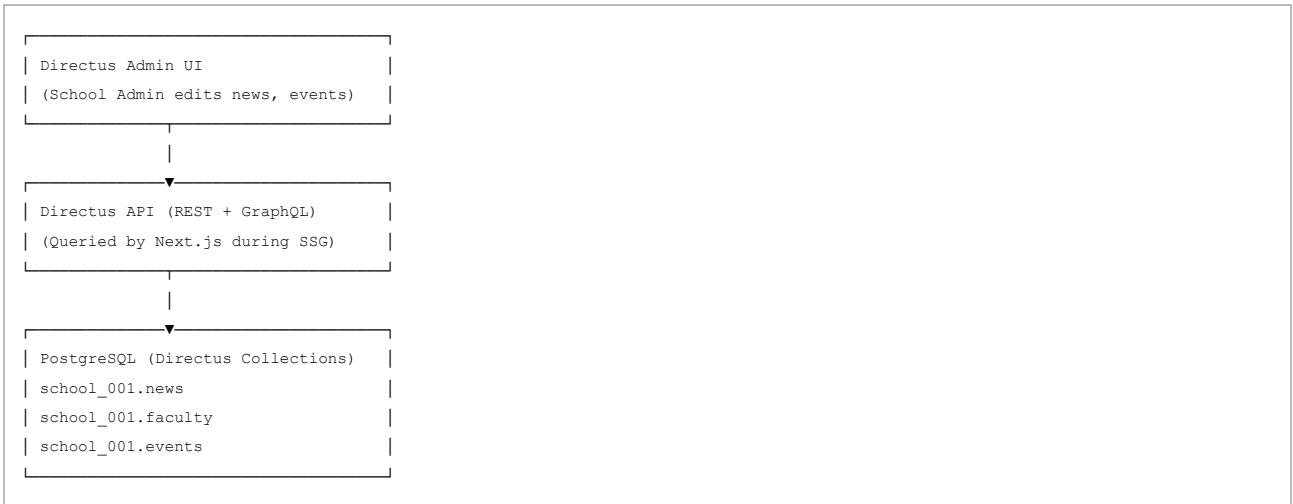
4.2 CMS Engine: Directus Integration

Role of Directus:

- Decouples content (news, faculty, events) from application code
- Provides no-code UI for school admins to manage public website content
- Exposes REST/GraphQL APIs for frontend to query

Architecture:

```
┌──────────────────────────────────┐
│ Directus Admin UI                │
│ (School Admin edits news, events)│
└──────────────────────────────────┘
                │
                ▼
┌──────────────────────────────────┐
│ Directus API (REST + GraphQL)    │
│ (Queried by Next.js during SSG)  │
└──────────────────────────────────┘
                │
                ▼
┌──────────────────────────────────┐
│ PostgreSQL (Directus Collections)│
│ school_001.news                  │
│ school_001.faculty               │
│ school_001.events                │
└──────────────────────────────────┘
```

Directus Collections per Tenant:

```
-- Directus auto-manages these via its API
school_001.news
  - id (UUID)
  - title (String)
  - content (Text, supports Markdown)
  - featured_image (File link)
  - published_at (DateTime)
  - seo_title, seo_description

school_001.faculty
  - id (UUID)
  - name (String)
  - bio (Text)
  - qualifications (String)
  - photo (File)
  - department_id (FK)

school_001.events
  - id (UUID)
  - title (String)
  - description (Text)
  - event_date (DateTime)
  - location (String)
  - max_attendees (Integer)
```

Next.js Integration (ISR for SEO):

```
// pages/news/[slug].tsx (Next.js)
import { ApolloClient, InMemoryCache, gql } from "@apollo/client";

const DIRECTUS_URL = process.env.DIRECTUS_API_URL; // Per-tenant Directus instance
const client = new ApolloClient({
  uri: `${DIRECTUS_URL}/graphql`,
  cache: new InMemoryCache(),
});

export async function getStaticProps({ params, locale }) {
  const { data } = await client.query({
    query: gql`
      query GetNews($id: ID!) {
        news(id: $id) {
          id
          title
          content
          featured_image { url }
          published_at
          seo_title
          seo_description
        }
      }
    `,
    variables: { id: params.slug },
  });

  return {
    props: { news: data.news, locale },
    revalidate: 3600, // ISR: regenerate every hour
  };
}

export default function NewsPage({ news }) {
  return (
    <>
      <Head>
        <title>{news.seo_title}</title>
        <meta name="description" content={news.seo_description} />
        <meta property="og:image" content={news.featured_image.url} />
      </Head>
      <div dangerouslySetInnerHTML={{ __html: markdownToHtml(news.content) }} />
    </>
  );
}
```

Multi-Language Support:

```
-- Directus translations
school_001.news_translations
  - id (UUID)
  - news_id (FK)
  - language_code (en, te, hi, etc.)
  - title (String, localized)
  - content (Text, localized)
```
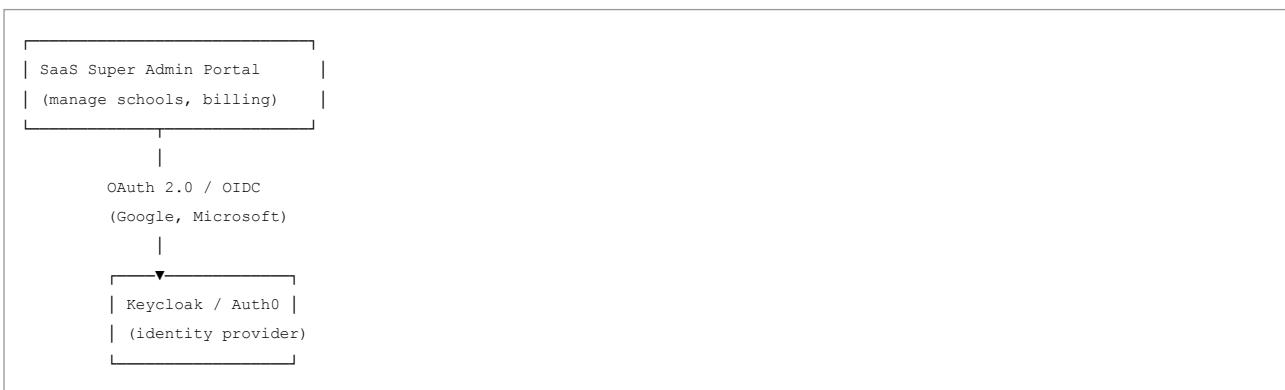
White-Labeling via Directus:

```
-- School-specific branding (stored in Directus)
school_001.site_config
  - school_name (String)
  - primary_color (String, e.g., #1E88E5)
  - secondary_color (String)
  - logo_url (File URL)
  - favicon_url (File URL)
  - footer_text (Text, editable by admin)
  - social_media_links (JSON)
    { "facebook": "...", "twitter": "...", "instagram": "..." }
```

4.3 Authentication & Authorization Hierarchy

Multi-Layer Auth Architecture

Layer 1: Super-Admin (SaaS Platform Admins)

```
 ┌─────────────────────────┐
 │ SaaS Super Admin Portal │
 │ (manage schools, billing) │
 └─────────────────────────┘
            │
     OAuth 2.0 / OIDC
     (Google, Microsoft)
            │
        ┌───▼──────────┐
        │ Keycloak / Auth0 │
        │ (identity provider) │
        └──────────────┘
```

Layer 2: School-Level Auth (School Admins, Teachers, Students)

```
 ┌─────────────────────────────┐
 │ School Portal (Mode A: Managed) │
 └─────────────────────────────┘
            │
        ┌───▼──────────┐
        │ JWT Authentication │
        │ (username/email │
        │  + password or  │
        │  SSO via SAML2) │
        └──────────────┘
            │
        ┌───▼──────────┐
        │ Access Token Issued │
        │ (claims include: │
        │  school_id,      │
        │  user_id,        │
        │  role,           │
        │  permissions)    │
        └──────────────┘
            │
        ┌───▼──────────┐
        │ Backend validates JWT │
        │ Extracts school_id   │
        │ Sets tenant context  │
        └──────────────┘
```

RBAC (Role-Based Access Control) Model:

```
// Tenant-scoped roles
@Entity
public class Role {
    UUID id;
    UUID schoolId;  // Tenant boundary
    String name;    // PRINCIPAL, TEACHER, STUDENT, PARENT, ADMIN
    Set<Permission> permissions;
    LocalDateTime createdAt;
}

@Entity
public class Permission {
    UUID id;
    String code;    // STUDENT_READ, GRADE_WRITE, FEE_APPROVE
    String description;
    Module module;  // STUDENT, TEACHER, EXAM, ADMIN
    PermissionLevel level;  // VIEW, CREATE, EDIT, DELETE
}

// User assignment to roles (school-scoped)
@Entity
public class UserRole {
    UUID id;
    UUID schoolId;  // Tenant boundary
    UUID userId;
    UUID roleId;
    LocalDateTime assignedAt;
}

// Example: Teacher can VIEW grades but not APPROVE payments
@Component
public class AuthorizationService {
    public boolean hasPermission(UUID schoolId, UUID userId, String permCode) {
        return userRoleRepo.findBySchoolIdAndUserId(schoolId, userId)
            .stream()
            .flatMap(ur -> ur.getRole().getPermissions().stream())
            .anyMatch(p -> p.getCode().equals(permCode));
    }
}
```

JWT Token Structure:

```
{
  "iss": "https://school-platform.com",
  "sub": "user-uuid-123",
  "school_id": "school-001",
  "email": "teacher@school.com",
  "role": "TEACHER",
  "permissions": ["STUDENT_READ", "ATTENDANCE_WRITE", "GRADE_READ"],
  "iat": 1733144229,
  "exp": 1733230629,
  "mfa_verified": true,
  "session_id": "session-abc-123"
}
```

Multi-Factor Authentication (MFA)

```
// Enforce MFA for sensitive roles (PRINCIPAL, ADMIN)
@Component
public class MfaService {
    public String initiateTotp(UUID userId) {
        String secret = generateSecret(); // Base32 encoded
        saveTotpSecret(userId, secret);
        return generateQrCode(secret); // Display to user's authenticator app
    }

    public boolean verifyTotp(UUID userId, String code) {
        String secret = getTotpSecret(userId);
        return TOTP.verify(secret, code, 30); // 30-sec window
    }

    public String initiateSms(UUID userId) {
        String phone = getUserPhone(userId);
        String code = generateOtp(6);
        sendSms(phone, "Your verification code: " + code);
        cacheOtp(userId, code, Duration.ofMinutes(5));
        return "OTP sent to " + maskPhone(phone);
    }
}
```

Session Management

```
@Configuration
public class SessionConfig {
    @Bean
    public SecurityFilterChain sessionChain(HttpSecurity http) {
        http.sessionManagement(session -> session
            .sessionFixationProtection(SessionFixationProtectionStrategy.MIGRAATE_SESSION)
            .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
            .maximumSessions(1)  // Only 1 active session per user
            .maxSessionsPreventsLogin(false)
            .expiredUrl("/login?expired")
        );
        return http.build();
    }
}

// Track sessions in Redis for distributed environments
@Component
public class SessionRegistry {
    public void recordSession(UUID schoolId, UUID userId, String sessionId) {
        redisTemplate.opsForValue().set(
            "session:school:" + schoolId + ":user:" + userId,
            sessionId,
            Duration.ofHours(24)
        );
    }

    public void invalidateOtherSessions(UUID userId, String currentSessionId) {
        redisTemplate.delete("session:*:user:" + userId);
        recordSession(schoolId, userId, currentSessionId);
    }
}
```
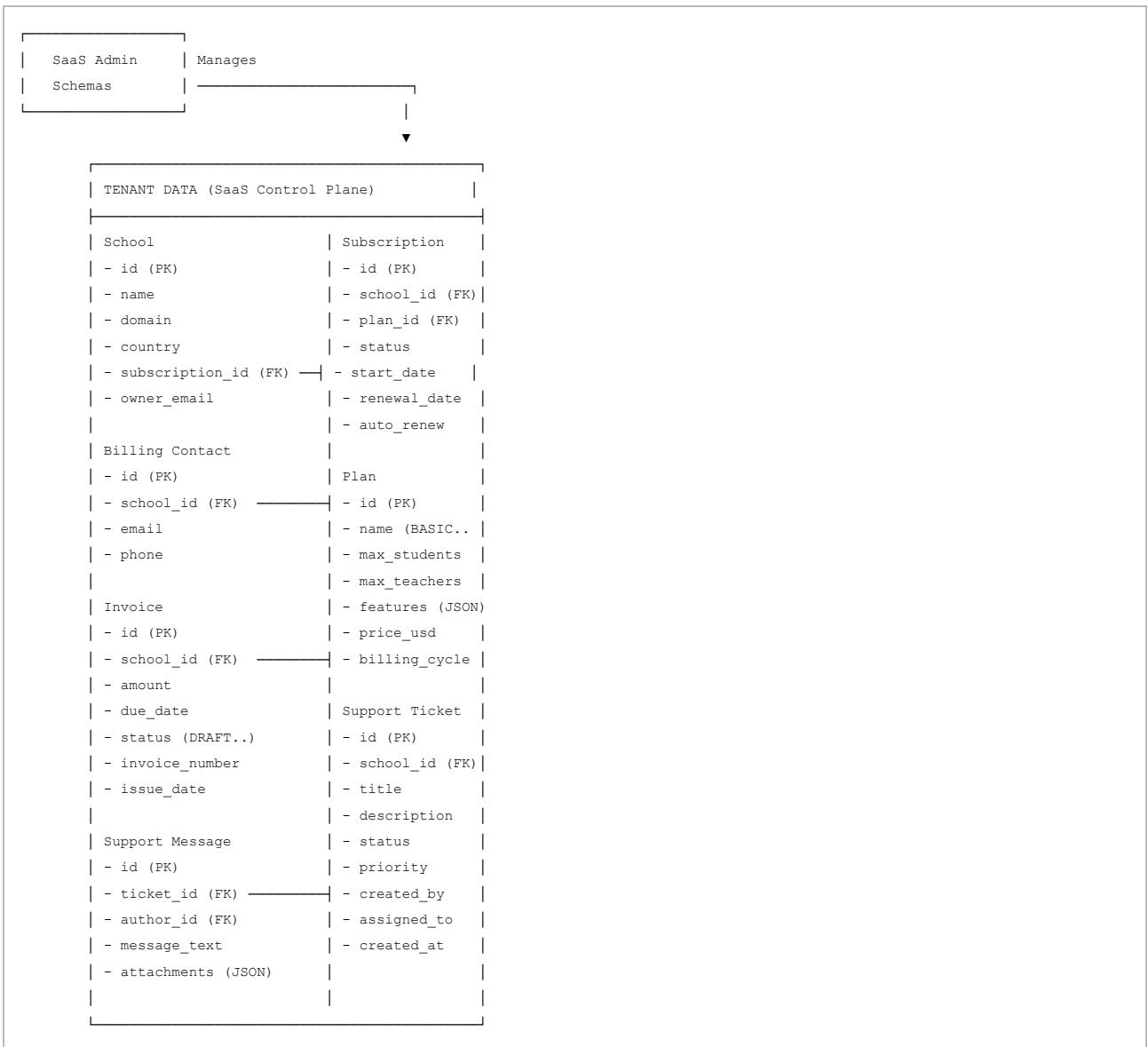
5. Data Model (Entity-Relationship Overview)

5.1 SaaS Platform Layer (SaaS DB - Shared)

```
┌──────────────┐
│ SaaS Admin   │ Manages
│ Schemas      │
└──────────────┘         ┐
                         │
                         │
                         ▼
    ┌─────────────────────────────────────────┐
    │ TENANT DATA (SaaS Control Plane)         │
    ├─────────────────────────────────────────┤
    │ School              │ Subscription       │
    │ - id (PK)           │ - id (PK)          │
    │ - name              │ - school_id (FK)   │
    │ - domain            │ - plan_id (FK)     │
    │ - country           │ - status           │
    │ - subscription_id (FK) ┤ - start_date     │
    │ - owner_email       │ - renewal_date     │
    │                     │ - auto_renew       │
    │ Billing Contact     │                    │
    │ - id (PK)           │ Plan               │
    │ - school_id (FK)  ────┤ - id (PK)        │
    │ - email             │ - name (BASIC.. │
    │ - phone             │ - max_students     │
    │                     │ - max_teachers     │
    │ Invoice             │ - features (JSON)  │
    │ - id (PK)           │ - price_usd        │
    │ - school_id (FK)  ────┤ - billing_cycle   │
    │ - amount            │                    │
    │ - due_date          │ Support Ticket     │
    │ - status (DRAFT..)  │ - id (PK)          │
    │ - invoice_number    │ - school_id (FK)   │
    │ - issue_date        │ - title            │
    │                     │ - description      │
    │ Support Message     │ - status           │
    │ - id (PK)           │ - priority         │
    │ - ticket_id (FK) ─────┤ - created_by      │
    │ - author_id (FK)    │ - assigned_to      │
    │ - message_text      │ - created_at       │
    │ - attachments (JSON)│                    │
    │                     │                    │
    └─────────────────────────────────────────┘
```

5.2 Tenant Layer (Per-Tenant DB/Schema)

Academic Hierarchy

```
School (root tenant entity)
  |
  ├ Classes/Divisions
  |   ├ Sections (A, B, C)
  |   |
  |   └ Subject Allocations
  |       └ Class-Teacher Mappings
  |
  ├ Students
  |   ├ Academic Records (grades, attendance)
  |   ├ Fee Payments
  |   └ Admissions (application → enrollment)
  |
  ├ Teachers
  |   ├ Leaves
  |   ├ Schedules (timetable)
  |   ├ Grading Rubrics
  |   └ Lesson Plans
  |
  └ Exams
      ├ Exam Schedules (conflict resolution)
      ├ Admit Cards
      ├ Question Banks
      ├ Results
      └ Transcripts
```

ER Diagram (Core Entities)

```sql
-- Students
CREATE TABLE students (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL REFERENCES schools(id),
    enrollment_number VARCHAR(50) UNIQUE,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    date_of_birth DATE,
    gender ENUM('M', 'F', 'OTHER'),
    email VARCHAR(255),
    phone VARCHAR(20),
    address_line_1 VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    postal_code VARCHAR(10),
    country VARCHAR(100),
    admission_date DATE,
    status ENUM('ACTIVE', 'INACTIVE', 'GRADUATED', 'TRANSFERRED'),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
    INDEX idx_school_status (school_id, status)
);


-- Classes (e.g., Class X-A)
CREATE TABLE classes (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL,
    academic_year INT,
    grade_level INT,
    section CHAR(1),
    class_teacher_id UUID,
    room_number VARCHAR(10),
    created_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
    CONSTRAINT fk_teacher FOREIGN KEY (class_teacher_id) REFERENCES teachers(id),
    UNIQUE(school_id, academic_year, grade_level, section)
);


-- Enrollments (Student-Class many-to-many with date range)
CREATE TABLE enrollments (
    id UUID PRIMARY KEY,
    student_id UUID NOT NULL,
    class_id UUID NOT NULL,
    school_id UUID NOT NULL,
    enrollment_date DATE,
    completion_date DATE,
    status ENUM('ONGOING', 'COMPLETED', 'DROPPED'),
    created_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES students(id),
    CONSTRAINT fk_class FOREIGN KEY (class_id) REFERENCES classes(id),
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
    UNIQUE(student_id, class_id)
);


-- Attendance
CREATE TABLE attendance (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL,
    student_id UUID NOT NULL,
    class_id UUID NOT NULL,
    attendance_date DATE,
    status ENUM('PRESENT', 'ABSENT', 'LEAVE', 'LATE'),
    marked_by_teacher_id UUID,
    remarks VARCHAR(500),
```

```sql
    created_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
    CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES students(id),
    CONSTRAINT fk_teacher FOREIGN KEY (marked_by_teacher_id) REFERENCES teachers(id),
    INDEX idx_attendance_date (school_id, attendance_date)
);

-- Grades
CREATE TABLE grades (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL,
    student_id UUID NOT NULL,
    exam_id UUID NOT NULL,
    subject_id UUID NOT NULL,
    marks_obtained DECIMAL(5, 2),
    total_marks DECIMAL(5, 2),
    grade_letter CHAR(1), -- A, B, C, D, F
    grade_point DECIMAL(3, 2),
    remarks VARCHAR(500),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
    CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES students(id),
    CONSTRAINT fk_exam FOREIGN KEY (exam_id) REFERENCES exams(id),
    INDEX idx_grades_exam (school_id, exam_id)
);

-- Exams
CREATE TABLE exams (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL,
    academic_year INT,
    exam_name VARCHAR(255),
    exam_type ENUM('MID_TERM', 'FINAL', 'UNIT_TEST', 'PRACTICAL'),
    start_date DATE,
    end_date DATE,
    result_published_date DATE,
    created_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id)
);

-- Exam Schedules (timetable)
CREATE TABLE exam_schedules (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL,
    exam_id UUID NOT NULL,
    subject_id UUID NOT NULL,
    scheduled_date DATE,
    start_time TIME,
    end_time TIME,
    room_number VARCHAR(10),
    max_marks INT,
    duration_minutes INT,
    created_at TIMESTAMP DEFAULT NOW(),
    CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
    CONSTRAINT fk_exam FOREIGN KEY (exam_id) REFERENCES exams(id),
    UNIQUE(exam_id, subject_id, scheduled_date)
);

-- Fees
CREATE TABLE fees (
    id UUID PRIMARY KEY,
    school_id UUID NOT NULL,
    student_id UUID NOT NULL,
    academic_year INT,
    fee_type ENUM('TUITION', 'TRANSPORT', 'UNIFORM', 'BOOKS', 'ACTIVITY'),
```

```sql
        amount DECIMAL(10, 2),
        due_date DATE,
        payment_status ENUM('PENDING', 'PARTIALLY_PAID', 'PAID', 'OVERDUE'),
        created_at TIMESTAMP DEFAULT NOW(),
        CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
        CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES students(id)
);


-- Fee Payments
CREATE TABLE fee_payments (
        id UUID PRIMARY KEY,
        school_id UUID NOT NULL,
        fee_id UUID NOT NULL,
        amount_paid DECIMAL(10, 2),
        payment_date TIMESTAMP,
        payment_method ENUM('CARD', 'NET_BANKING', 'UPI', 'CHECK', 'CASH'),
        transaction_id VARCHAR(255),
        payment_gateway_ref VARCHAR(255),
        status ENUM('SUCCESS', 'FAILED', 'PENDING'),
        created_at TIMESTAMP DEFAULT NOW(),
        CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
        CONSTRAINT fk_fee FOREIGN KEY (fee_id) REFERENCES fees(id),
        INDEX idx_payments_status (school_id, payment_status)
);


-- Teachers
CREATE TABLE teachers (
        id UUID PRIMARY KEY,
        school_id UUID NOT NULL,
        employee_id VARCHAR(50) UNIQUE,
        first_name VARCHAR(255),
        last_name VARCHAR(255),
        email VARCHAR(255),
        phone VARCHAR(20),
        date_of_joining DATE,
        qualifications VARCHAR(500),
        specializations VARCHAR(500),
        employment_status ENUM('ACTIVE', 'ON_LEAVE', 'RESIGNED', 'RETIRED'),
        created_at TIMESTAMP DEFAULT NOW(),
        CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id)
);


-- Subjects
CREATE TABLE subjects (
        id UUID PRIMARY KEY,
        school_id UUID NOT NULL,
        subject_code VARCHAR(10),
        subject_name VARCHAR(255),
        description TEXT,
        credits INT,
        created_at TIMESTAMP DEFAULT NOW(),
        CONSTRAINT fk_school FOREIGN KEY (school_id) REFERENCES schools(id),
        UNIQUE(school_id, subject_code)
);
```

Tenant Context Propagation

```
// Every query operation includes tenant context
@Component
public class TenantContext {
    private static final ThreadLocal<UUID> currentSchoolId = new ThreadLocal<>();

    public static void setCurrentSchool(UUID schoolId) {
        currentSchoolId.set(schoolId);
    }

    public static UUID getCurrentSchool() {
        return currentSchoolId.get();
    }

    public static void clear() {
        currentSchoolId.remove();
    }
}

// Interceptor to set context from JWT
@Component
public class TenantContextInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {
        String schoolId = request.getHeader("X-School-ID");
        if (schoolId != null) {
            TenantContext.setCurrentSchool(UUID.fromString(schoolId));
        }
        return true;
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response,
                                Object handler, Exception ex) {
        TenantContext.clear();
    }
}

// Repository automatically filters by tenant
@Repository
public class StudentRepository extends JpaRepository<Student, UUID> {
    @Override
    default List<Student> findAll() {
        UUID schoolId = TenantContext.getCurrentSchool();
        return findBySchoolId(schoolId);
    }
}
```

6. Non-Functional Requirements (NFRs)

6.1 Scalability

Handling Result-Day Traffic Spikes

Exam Result Release Scenario:

- 50 schools × 500 students/school = 25K concurrent users checking results
- Peak QPS: ~2500 queries/second (concurrent check-ins)

Scaling Strategy:

```
1. Pre-compute Results (Batch Job)
   ├─ T-1 hour: Result data cached in Valkey
   ├─ Key: results:exam_{exam_id}:student_{student_id}
   └─ TTL: 24 hours


2. Database Query Optimization
   ├─ Add index on (exam_id, student_id, school_id)
   └─ Use read replicas for result queries


3. API Rate Limiting (per student)
   ├─ Max 10 requests/minute
   └─ Graceful 429 response with retry-after header


4. Horizontal Scaling
   ├─ Docker Swarm auto-scale to 10 replicas during exams
   ├─ Trigger: CPU > 70% for 2 minutes
   └─ Traefik load-balances across replicas


5. Frontend Optimization
   ├─ Client-side retry logic with exponential backoff
   ├─ Progressive loading (show cached partial results first)
   └─ WebSocket for real-time result availability notification
```

Docker Swarm Auto-Scaling:

```
deploy:
  replicas: 3  # Default
  autoscaling:
    targets:
      - cpu: 70
        replicas: 10
      - memory: 80
        replicas: 8
  update_config:
    parallelism: 2  # Update 2 replicas at a time
    delay: 10s
```

Result Caching Pipeline:

```
@Service
public class ResultProcessingService {

    @Scheduled(cron = "0 0 12 * * ?") // Midnight IST
    public void preComputeResults() {
        List<Exam> readyExams = examRepo.findByStatusAndPublishDate(READY, TODAY);

        for (Exam exam : readyExams) {
            List<Grade> grades = gradeRepo.findByExamId(exam.getId());

            for (Grade grade : grades) {
                String cacheKey = "results:exam:" + exam.getId() +
                                  ":student:" + grade.getStudentId();

                ResultDTO dto = new ResultDTO(grade, exam);
                // Cache with 24-hour expiry
                cache.put(cacheKey, dto, Duration.ofHours(24));

                // Publish WebSocket event: result available
                resultEventPublisher.publishResultAvailable(grade.getStudentId());
            }
        }
    }

    public ResultDTO getStudentResult(UUID examId, UUID studentId) {
        UUID schoolId = TenantContext.getCurrentSchool();
        String cacheKey = "results:exam:" + examId + ":student:" + studentId;

        // Try cache first (99% hit on result day)
        return cache.getOrCompute(cacheKey, () ->
            gradeRepo.findByExamIdAndStudentIdAndSchoolId(examId, studentId, schoolId)
                .map(ResultDTO::new)
                .orElse(null)
        );
    }
}
```

Load Testing Metrics

```
# Using JMeter/Gatling
# Simulating 25K concurrent users checking results

Scenario: Get Student Result
- Ramp up: 5000 users/minute
- Sustain: 25K users for 30 minutes
- Ramp down: 5000 users/minute

Results (Target):
- P95 latency: < 200ms (cached)
- P99 latency: < 500ms (DB query)
- Error rate: < 0.1%
- Throughput: 2500+ QPS
```

6.2 Security

Data Encryption

At-Rest Encryption:

```
-- Sensitive fields encrypted in database
ALTER TABLE students
  ADD COLUMN ssn_encrypted BYTEA,
  ADD COLUMN aadhar_encrypted BYTEA;


-- Transparent encryption via PostgreSQL pgcrypto
CREATE TABLE students (
    ssn_encrypted TEXT DEFAULT encode(
        encrypt(ssn::TEXT, 'school_key'::BYTEA, 'aes'),
        'hex'
    )
);


-- Or use column-level encryption at application layer
@Column(columnDefinition = "BYTEA")
@Convert(converter = EncryptedStringConverter.class)
private String socialSecurityNumber;
```

In-Transit Encryption:

```
# TLS 1.3 enforced
server:
  ssl:
    enabled: true
    key-store: /secrets/keystore.jks
    key-store-password: ${KEY_STORE_PASS}
    key-store-type: PKCS12
    protocol: TLSv1.3
    ciphers: TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256
```

GDPR Compliance

Right to Erasure ("Forgotten"):

```
@Service
public class GdprErasureService {

    public void eraseStudentData(UUID schoolId, UUID studentId) {
        // Soft delete: mark as erased
        Student student = studentRepo.findById(studentId);
        student.setErasedAt(LocalDateTime.now());
        student.setFirstName("ERASED");
        student.setLastName("ERASED");
        student.setEmail("erased@school-platform.com");
        // Retain minimal data for financial/legal compliance

        // Cascade delete: grades, attendance, etc.
        gradeRepo.deleteByStudentIdAndSchoolId(studentId, schoolId);
        attendanceRepo.deleteByStudentIdAndSchoolId(studentId, schoolId);

        // Audit log erasure request
        auditLog.record(schoolId, studentId, "DATA_ERASED", LocalDateTime.now());
    }
}
```

Data Localization:

- EU tenant data → AWS eu-west-1 (Ireland)
- India tenant data → AWS ap-south-1 (Mumbai)
- US tenant data → AWS us-east-1 (N. Virginia)

COPPA Compliance (Children's Online Privacy Protection)

Applies to students under 13 in US schools.

```
@Service
public class CoppaComplianceService {

    public void registerStudentUnder13(UUID schoolId, UUID studentId, StudentDTO student) {
        // Require parental consent
        if (student.getDateOfBirth().isAfter(LocalDate.now().minusYears(13))) {
            ParentalConsent consent = new ParentalConsent();
            consent.setStudentId(studentId);
            consent.setConsentType(COPPA_PARENTAL_CONSENT);
            consent.setStatus(PENDING);
            consent.setExpiresAt(LocalDateTime.now().plusDays(30));

            consentRepo.save(consent);

            // Send email to parent for consent
            emailService.sendParentalConsentRequest(
                student.getParentEmail(),
                consent.getId()
            );

            throw new CoppaComplianceException("Parental consent required");
        }
    }

    public void storeParentalConsent(UUID consentId, String parentEmail, boolean approved) {
        ParentalConsent consent = consentRepo.findById(consentId);
        consent.setApproved(approved);
        consent.setParentEmail(parentEmail);
        consent.setConsentedAt(LocalDateTime.now());
        consentRepo.save(consent);

        if (!approved) {
            // Student account disabled
            studentRepo.findById(consent.getStudentId()).ifPresent(s -> {
                s.setStatus(StudentStatus.COPPA_CONSENT_REJECTED);
                studentRepo.save(s);
            });
        }
    }
}
```

API Security Headers

```
@Configuration
public class SecurityHeadersConfig {

    @Bean
    public SecurityFilterChain securityHeaders(HttpSecurity http) {
        http.headers(headers -> headers
            .contentSecurityPolicy("default-src 'self'; script-src 'self' cdn.jsdelivr.net")
            .xssProtection()
            .frameOptions().deny()
            .httpStrictTransportSecurity().includeSubDomains().maxAgeInSeconds(63072000)
            .referrerPolicy(ReferrerPolicy.STRICT_ORIGIN_WHEN_CROSS_ORIGIN)
        );
        return http.build();
    }
}
```

6.3 Observability

Logging Strategy

Multi-Level Logging:

```
@Component
public class TenantAwareLogger {
    private static final Logger logger = LoggerFactory.getLogger(TenantAwareLogger.class);

    public void logBusinessEvent(String eventType, Map<String, Object> context) {
        UUID schoolId = TenantContext.getCurrentSchool();

        MDC.put("school_id", schoolId.toString());
        MDC.put("timestamp", Instant.now().toString());
        MDC.put("event_type", eventType);

        logger.info("BUSINESS_EVENT", context);

        MDC.clear();
    }
}

// Log structure (Loki-compatible JSON)
{
  "timestamp": "2025-12-02T10:30:00Z",
  "level": "INFO",
  "logger": "com.schoolsaas.StudentService",
  "school_id": "school-001",
  "user_id": "user-123",
  "event_type": "GRADE_PUBLISHED",
  "exam_id": "exam-456",
  "student_count": 250,
  "duration_ms": 2450,
  "message": "Published exam results for 250 students"
}
```

Centralized Log Aggregation (Loki):

```
# promtail-config.yaml
scrape_configs:
  - job_name: spring-backend
    docker_sd_configs:
      - host: unix:///var/run/docker.sock
    relabel_configs:
      - source_labels: ['__meta_docker_container_label_app']
        target_label: 'app'
      - source_labels: ['__meta_docker_container_label_school']
        target_label: 'school_id'

  - job_name: directus
    docker_sd_configs:
      - host: unix:///var/run/docker.sock

clients:
  - url: http://loki:3100/loki/api/v1/push
```

Metrics & Monitoring (Prometheus)

```
# prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'spring-backend'
    static_configs:
      - targets: ['localhost:8080']
    metrics_path: '/actuator/prometheus'

  - job_name: 'postgres'
    static_configs:
      - targets: ['localhost:5432']

  - job_name: 'valkey'
    static_configs:
      - targets: ['localhost:6379']
```

Custom Metrics (Spring Boot Actuator):

```
@Component
public class SchoolSaasMetrics {
    private final MeterRegistry meterRegistry;

    @PostConstruct
    public void registerMetrics() {
        // Active student dashboard sessions
        Gauge.builder("school.active_sessions", this::getActiveSessions)
            .tag("type", "student")
            .register(meterRegistry);

        // Grade publishing latency
        Timer.builder("exam.grade_publish_latency")
            .publishPercentiles(0.5, 0.95, 0.99)
            .register(meterRegistry);

        // Fee payment success rate
        Counter.builder("fee.payment_success_total")
            .description("Total successful fee payments")
            .tag("payment_method", "card")
            .register(meterRegistry);
    }

    private double getActiveSessions() {
        return sessionRegistry.getActiveSessionCount();
    }
}
```

Grafana Dashboards (Pre-built):

Dashboard 1: **SaaS Control Plane**

- Active schools
- Subscription revenue trend
- Support ticket SLA compliance
- System health (uptime %)

Dashboard 2: **Per-Tenant Performance**

- API response times (P50, P95, P99)
- Database query performance
- Cache hit rate
- Concurrent users

Dashboard 3: **Security & Compliance**

- Failed login attempts

- Data access audit logs
- GDPR erasure requests
- API key rotation status

Distributed Tracing (Jaeger)

```java
@Configuration
public class TracingConfig {
    @Bean
    public JaegerTracer jaegerTracer() {
        return new Configuration("school-saas")
            .withSampler(new Configuration.SamplerConfiguration()
                .withType("const")
                .withParam(1)
            )
            .withReporter(new Configuration.ReporterConfiguration()
                .withLogSpans(true)
                .withAgentHost("jaeger-agent")
                .withAgentPort(6831)
            )
            .getTracer();
    }
}


// Tracing exam result publishing (async flow)
@Service
public class ResultProcessingService {

    @Traced(operationName = "publish_exam_results")
    public void publishResults(UUID examId) {
        Span span = jaegerTracer.startSpan("publish_results");
        try {
            // Pre-compute results (1.5s)
            preComputeResults(examId);

            // Publish notifications (2s)
            publishNotifications(examId);

            // Generate transcripts (3s)
            generateTranscripts(examId);
        } finally {
            span.finish();
        }
    }
}

// Jaeger UI shows: exam results published in 6.5s with 3 async spans
```

7. Deployment Strategy

7.1 CI/CD Pipeline (GitHub Actions + ArgoCD)

```yaml
# .github/workflows/deploy.yml
name: Deploy School SaaS

on:
  push:
    branches: [main, staging]
    paths:
      - 'src/**'
      - 'docker/**'

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build & Test
        run: |
          ./mvnw clean package -DskipTests=false

      - name: Build Docker Image
        run: |
          docker build -t schoolsaas/backend:${{ github.sha }} .
          docker push ghcr.io/schoolsaas/backend:${{ github.sha }}

      - name: Update Manifest
        run: |
          sed -i 's|IMAGE_TAG|${{ github.sha }}|g' k8s/deployment.yaml

      - name: Push to GitOps Repo
        run: |
          git -C ../gitops-config pull
          cp k8s/deployment.yaml ../gitops-config/
          git -C ../gitops-config commit -am "Deploy ${{ github.sha }}"
          git -C ../gitops-config push

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - name: Deploy via ArgoCD
        run: |
          argocd app sync school-saas-prod --prune
          argocd app wait school-saas-prod
```

7.2 Canary Deployment Per Tenant

```
# docker-service-update.yml (for critical updates)
version: '3.9'
services:
  school-backend:
    image: schoolsaas/backend:v2.1.0
    deploy:
      update_config:
        parallelism: 1         # Update 1 replica at a time
        delay: 30s             # Wait 30s between updates
        failure_action: rollback
        monitor: 30s           # Monitor for 30s post-update
      replicas: 5
      # Canary: 10% of traffic to new version
      placement:
        constraints:
          - node.labels.version == v2.0.0 OR
            node.labels.version == v2.1.0
```

Gradual Rollout for Custom Plans:

```
# Tenant-specific deployment (Database-per-Tenant plans)
# New version deployed to 1 school first, then others

Day 1: Deploy to "Pilot School"
  → Run on v2.1.0
  → Monitor error rates, latency
  → Collect feedback

Days 2-5: Deploy to 20% of Custom schools
  → If issues detected, rollback
  → Otherwise continue

Week 2: Deploy to remaining 80%
```

8. Risk Mitigation & Contingency

Risk	Impact	Mitigation
Multi-tenant data breach	CRITICAL	Row-level security + encryption + audit logs + immutable backups
Exam result delay	HIGH	Caching + read replicas + graceful degradation + manual fallback
Database corruption	CRITICAL	WAL-based backups every 1 hour + test restores weekly
Payment gateway downtime	MEDIUM	Queue fee submissions; retry with exponential backoff
Directus CMS unavailable	MEDIUM	Cache CMS responses in Valkey; fall back to static HTML
Docker Swarm node failure	MEDIUM	Multi-zone deployment + auto-failover + PV replication

9. Cost Optimization Roadmap

Year 1 (MVP): $50K/month

- 10 schools (Basic) @ $50/school = $500/month
- 5 schools (Premium) @ $150/school = $750/month
- SaaS Portal + support staff: $49K/month
- Total revenue: $1,250/month (startup loss: -$48,750/month)

Year 2: $15K/month (break-even targeted)

- 100 schools (Mixed plans) @ avg $150/school = $15K/month
- Infrastructure optimized: $8K/month
- Headcount reduced via automation: $7K/month

Year 3+: Profitable

- 500+ schools
```

- Infrastructure leveraged: $20K/month
- Revenue: $75K/month
- Net margin: ~60%

---

# 10. Success Criteria & KPIs

| Metric | Target | Timeline |
|---|---|---|
| Platform Uptime | 99.9% (43 min downtime/month) | Continuous |
| API Latency (P95) | < 200ms | Month 3 |
| User Onboarding Time | < 2 hours (automated setup) | Month 6 |
| Time to First Value | School admin logs in → sees class list within 5 min | Month 1 |
| Support Ticket Resolution | 95% within 24 hours | Month 2 |
| User Adoption Rate | 70% of school staff use system | Month 12 |
| NPS Score | > 50 | Year 2 |
| Tenant Churn Rate | < 5% annually | Year 2 |

---

# Conclusion

This HLD document provides a **production-ready architecture** for a multi-tenant School Management SaaS platform. The **Spring Boot Modulith with Docker Swarm deployment** balances cost-efficiency, scalability, and operational simplicity. The **hybrid multi-tenancy strategy** (shared schema for standard plans, dedicated databases for enterprise) ensures profitability while maintaining premium security.

Key differentiators: □ **100% FOSS stack** (PostgreSQL, Valkey, Traefik, Prometheus)
□ **White-label ready** via Directus CMS integration
□ **Hybrid delivery modes** (Managed + Headless API)
□ **Enterprise-grade security** (RLS, encryption, audit logs)
□ **Scalable to 50K concurrent users** (result day traffic)
□ **Clear migration path** from monolith → microservices if needed

This architecture positions the platform for rapid market entry while preserving the ability to scale to millions of students globally.

---

**Document Version:** 1.0
**Last Updated:** December 2, 2025
**Approved By:** Chief Software Architect