LMS Platform - Product Requirements Document

Overall Project Overview

Executive Summary

The **LMS Platform** is an enterprise-grade Learning Management System designed for educational institutions (colleges, universities, and multi-campus organizations). Built on a microservices architecture, it provides comprehensive identity management, course delivery, assessment, collaboration, and analytics capabilities. The platform currently has a production-ready authentication foundation with edge security (Kong Gateway + Vault) and is expanding to deliver full LMS functionality.

Target Audience

Primary Users:

- **Students** (10,000+ concurrent users expected): Course enrollment, content access, assignment submission, collaboration
- **Teachers/Faculty** (500-1000 users): Course creation, grading, attendance tracking, student interaction
- **Administrative Staff**:
  - **HODs (Heads of Department)**: Department-level course/faculty management
  - **Deans**: College-level oversight, policy enforcement
  - **Non-Teaching Staff**: Administrative support, data entry
- **College Administrators**: Multi-department management, reporting
- **Organization Admins**: Multi-college platform governance

Secondary Users:

- **Parents/Guardians**: Progress monitoring (future phase)
- **External Examiners**: Assessment validation (future phase)

- **System Administrators**: Platform maintenance, monitoring

Key Goals

1. **User Adoption**: Achieve 90% active user adoption within first academic year post-launch
2. **Operational Efficiency**: Reduce administrative overhead by 40% through automation (bulk imports, auto-grading, attendance automation)
3. **Academic Performance**: Improve student engagement metrics by 30% via interactive content and real-time feedback
4. **System Reliability**: Maintain 99.5% uptime during academic sessions
5. **Security Compliance**: Zero security breaches; full GDPR/FERPA compliance
6. **Scalability**: Support 50,000+ users across 100+ colleges without performance degradation

Success Metrics

Metric Category	KPI	Target	Measurement Method
Adoption	Active daily users	>75% of enrolled students	Analytics dashboard
Performance	API response time	<500ms (p95)	Prometheus metrics
Reliability	System uptime	>99.5%	CloudWatch/ Grafana
Engagement	Course completion rate	>85%	Course analytics
Efficiency	Admin task time reduction	>40%	Time-tracking surveys
Security	Security incidents	0 critical	Security audit logs
Scalability	Concurrent users supported	10,000+	Load testing (Artillery)

Metric Category	KPI	Target	Measurement Method
Data Integrity	Data loss incidents	0	Backup validation

Technology Stack (Current + Planned)

Backend:

- Node.js + Express.js (primary), NestJS (select services)
- TypeScript
- Prisma ORM + PostgreSQL (RDS/Aurora planned)
- MongoDB (for unstructured content - planned)
- Redis (Elasticache) - caching, sessions, pub/sub

Messaging & Events:

- Apache Kafka (AWS MSK planned)
- Socket.IO (real-time features)

Security:

- HashiCorp Vault (secrets management)
- PASETO tokens (authentication)
- Kong API Gateway (routing, rate limiting, JWT validation)
- Nginx (reverse proxy)

Storage:

- AWS S3 (media, documents)
- PostgreSQL (structured data)
- Elasticsearch (search indexing - planned)

DevOps & Infrastructure:

- Docker + Docker Compose
- AWS EKS/ECS (planned)
- Terraform (IaC - planned)
- GitHub Actions (CI/CD - planned)

Monitoring & Observability:

- Prometheus + Grafana (planned)
- Elasticsearch + Kibana (planned)
- OpenTelemetry (tracing - planned)
- CloudWatch (AWS - planned)

Frontend (Planned):

- Next.js/React
- TypeScript
- TailwindCSS
- Redux/Zustand (state management)

Current State Assessment

✅ **Completed (Production-Ready):**

1. **Edge Authentication Architecture**: Kong Gateway with custom Lua plugin for PASETO/Vault validation
2. **Secret Management**: Vault policies for auth-service and Kong plugin
3. **Database Schema**: Complex academic hierarchy (Organization → College → Branch → Batch → Section → Student)
4. **Multi-Role Authentication**: Login/Register/Verify for 7 user types (Organization, College, Dean, HOD, Teacher, Student, Staff)
5. **Notification Infrastructure**: Kafka-based email notifications (OTP, welcome, password reset)
6. **Infrastructure Service**: Kafka topic management and administration

In Progress (Partial Implementation):

1. **Single Device Login**: Schema exists, enforcement logic needs completion
2. **Bulk Data Ingestion**: Documentation exists, needs async queue implementation (Kafka workers)
3. **API Documentation**: Swagger/OpenAPI partially configured

🟤 **Pending (Roadmap):**

1. **Frontend Applications**: No UI code (Super Admin, College Portal, Student LMS)

2. **Core LMS Services**: Course management, attendance, content delivery, assessments
3. **Monitoring Stack**: Prometheus/Grafana/Elasticsearch setup
4. **Cloud Deployment**: AWS infrastructure (currently Docker Compose only)
5. **Advanced Features**: Analytics, AI/ML, mobile apps, integrations

Assumptions

1. **Infrastructure**: AWS will be the primary cloud provider for production deployment
2. **User Base**: Initial rollout to 5-10 pilot colleges (5,000-10,000 users) before full scale
3. **Network**: Users have stable internet (minimum 2 Mbps for video content)
4. **Devices**: Students/faculty have access to laptops/tablets/smartphones (responsive design required)
5. **Data Migration**: Existing colleges will provide clean CSV/JSON data for bulk imports
6. **Compliance**: Educational institutions will provide GDPR/FERPA compliance requirements upfront
7. **Integration**: Third-party integrations (payment gateways, video platforms) will have stable APIs
8. **Content**: Faculty will create/migrate course content progressively (not all upfront)

Risks & Mitigation

Risk	Impact	Probability	Mitigation Strategy
Kafka message loss	High	Medium	Implement message persistence, replication factor 3, consumer offset management
Vault unavailability	Critical	Low	Multi-node Vault cluster, automated unsealing, backup policies
Database performance degradation	High	Medium	Connection pooling, read replicas, query optimization, caching layer
		High	

Risk	Impact	Probability	Mitigation Strategy
Single device login race conditions	Medium		Redis-based distributed locking, Kafka pub/sub for session invalidation
Bulk import timeouts	Medium	High	Async processing with Kafka workers, progress tracking, chunked uploads
GDPR compliance violations	Critical	Low	Data encryption at rest/transit, audit logging, right-to-deletion workflows
Frontend-backend version mismatch	Medium	Medium	API versioning (/api/v1, /api/v2), backward compatibility, feature flags
Third-party service outages	Medium	Medium	Circuit breakers, fallback mechanisms, SLA monitoring
Insufficient load testing	High	Medium	Continuous load testing with Artillery, chaos engineering (Phase 20+)
Skill gaps in team	Medium	Medium	Training programs, documentation, code reviews, pair programming

Compliance & Regulatory Requirements

1. **GDPR (General Data Protection Regulation)**:
    - Right to access, rectification, erasure
    - Data portability
    - Consent management
    - Breach notification (72 hours)
2. **FERPA (Family Educational Rights and Privacy Act)**:
    - Student record privacy
    - Parental access controls
    - Third-party disclosure restrictions

3. **Accessibility**:
    - WCAG 2.1 Level AA compliance
    - Screen reader compatibility
    - Keyboard navigation
    - Color contrast ratios
4. **Security Standards**:
    - OWASP Top 10 mitigation
    - SOC 2 Type II (future)
    - ISO 27001 alignment

Project Phases Overview

The project is divided into **25 micro-phases**, each delivering incremental value:

Foundation & Security (Phases 1-5)

- Phase 1: Auth Service Hardening
- Phase 2: Single Device Login Enforcement
- Phase 3: Bulk Import Infrastructure
- Phase 4: Monitoring & Observability Foundation
- Phase 5: API Gateway Enhancement

Core LMS Features (Phases 6-12)

- Phase 6: Course Management Service
- Phase 7: Student Enrollment & Allocation
- Phase 8: Content Management System (CMS)
- Phase 9: Attendance Service
- Phase 10: Assignment & Submission Service
- Phase 11: Grading & Assessment Engine
- Phase 12: Real-time Chat Service

Advanced Features (Phases 13-18)

- Phase 13: Search Service (Elasticsearch)
- Phase 14: Analytics & Reporting Service
- Phase 15: Notification Enhancement (SMS, Push)
- Phase 16: Timetable & Scheduling Service

- Phase 17: Library Management Integration
- Phase 18: Payment & Fee Management

Frontend & UX (Phases 19-21)

- Phase 19: Super Admin Portal (Next.js)
- Phase 20: College Management Portal
- Phase 21: Student LMS Portal

Production & Scale (Phases 22-25)

- Phase 22: AWS Infrastructure (Terraform)
- Phase 23: CI/CD Pipeline (GitHub Actions)
- Phase 24: Mobile Applications (React Native)
- Phase 25: AI/ML Features (Recommendations, Proctoring)

Rollout Plan

Alpha Testing (Phases 1-10)

- Internal team testing
- Synthetic data
- Performance benchmarking
- Security audits

Beta Testing (Phases 11-18)

- 2-3 pilot colleges
- 500-1000 real users
- Feedback loops (bi-weekly)
- Bug bounty program

Limited Release (Phases 19-21)

- 5-10 colleges
- 5,000-10,000 users
- Gradual feature rollout
- A/B testing for UX

General Availability (Phases 22-25)

- All partner colleges
- 50,000+ users
- Full feature set
- 24/7 support

Document Structure

This PRD is organized into multiple files:

- `00-OVERVIEW.md` (this file): Project overview, goals, metrics
- `01-PHASES-01-05.md`: Phases 1-5 (Foundation & Security)
- `02-PHASES-06-10.md`: Phases 6-10 (Core LMS - Part 1)
- `03-PHASES-11-15.md`: Phases 11-15 (Core LMS - Part 2 & Advanced)
- `04-PHASES-16-20.md`: Phases 16-20 (Advanced Features & Frontend)
- `05-PHASES-21-25.md`: Phases 21-25 (Production & Scale)
- `06-APPENDIX.md`: Data models, API contracts, glossary

Document Version: 1.0
Last Updated: December 2025
Owner: Product & Engineering Team
Status: Living Document (Updated per phase completion)

LMS Platform PRD - Table of Contents

📚 Complete Documentation Index

This document provides a comprehensive index of all PRD documents, sections, and key topics for easy navigation.

Document Structure

Core Documents (7 Phase Documents + 3 Supporting Documents)

Document	File	Pages	Description
Overview	00-OVERVIEW.md	~15	Project overview, goals, metrics, tech stack
Phases 1-5	01-PHASES-01-05.md	~40	Foundation & Security
Phases 6-9	02-PHASES-06-09.md	~35	Core LMS Part 1
Phases 10-12	03-PHASES-10-12.md	~30	Core LMS Part 2
Phases 13-18	04-PHASES-13-18.md	~25	Advanced Features
Phases 19-20		~35	Frontend Portals

Document	File	Pages	Description
	05-PHASES-19-20.md		
Phases 21-25	06-PHASES-21-25.md	~40	Production & AI/ML
Appendix	07-APPENDIX.md	~20	Data models, API contracts, glossary
Phase Breakdown	PHASE-BREAKDOWN.md	~10	Quick reference for all phases
README	README.md	~5	Navigation guide

Total: ~255 pages of comprehensive documentation

Detailed Table of Contents

00-OVERVIEW.md

3. Current State Assessment

- 3.1 Completed Components
- 3.2 In-Progress Features
- 3.3 Pending Features

4. Assumptions & Risks

- 4.1 Project Assumptions
- 4.2 Risk Analysis & Mitigation
- 4.3 Compliance Requirements

5. Project Phases Overview

- 5.1 Foundation & Security (Phases 1-5)
- 5.2 Core LMS Features (Phases 6-12)
- 5.3 Advanced Features (Phases 13-18)
- 5.4 Frontend & UX (Phases 19-21)
- 5.5 Production & Scale (Phases 22-25)

6. Rollout Plan

- 6.1 Alpha Testing
- 6.2 Beta Testing
- 6.3 Limited Release
- 6.4 General Availability

01-PHASES-01-05.md - Foundation & Security

Phase 1: Auth Service Hardening

- 1.1 Phase Goals
- 1.2 User Stories (4 stories)
- 1.3 Functional Requirements
    - FR1.1: Audit Logging
    - FR1.2: Account Lockout
    - FR1.3: Password Policy
    - FR1.4: Token Refresh Enhancement

        - FR1.5: Rate Limiting
        - FR1.6: Error Standardization
    - 1.4 Non-Functional Requirements
    - 1.5 Technical Architecture
        - Data Models (Prisma)
        - Kafka Topics
        - Redis Keys
    - 1.6 Dependencies & Tech Stack
    - 1.7 Verification Steps (8 tests)

Phase 2: Single Device Login Enforcement

- 2.1 Phase Goals
- 2.2 User Stories (4 stories)
- 2.3 Functional Requirements
    - FR2.1: Device Fingerprinting
    - FR2.2: Session Management
    - FR2.3: Real-time Invalidation
    - FR2.4: Race Condition Handling
    - FR2.5: Device Management API
    - FR2.6: New Device Notifications
- 2.4 Non-Functional Requirements
- 2.5 Technical Architecture
    - Data Models
    - Redis Data Structures
    - Kong Gateway Update
- 2.6 Verification Steps (8 tests)

Phase 3: Bulk Import Infrastructure

- 3.1 Phase Goals
- 3.2 User Stories (4 stories)
- 3.3 Functional Requirements
    - FR3.1: File Upload & Validation
    - FR3.2: Asynchronous Job Processing
    - FR3.3: Worker Service
    - FR3.4: Progress Tracking
    - FR3.5: Error Reporting

        - FR3.6: Rollback Mechanism
        - FR3.7: Notification Integration
    - 3.4 Technical Architecture
        - New Service Structure
        - Data Models
        - Kafka Topics
        - S3 Bucket Structure
    - 3.5 Verification Steps (8 tests)

Phase 4: Monitoring & Observability Foundation

- 4.1 Phase Goals
- 4.2 User Stories (4 stories)
- 4.3 Functional Requirements
    - FR4.1: Prometheus Metrics
    - FR4.2: Grafana Dashboards
    - FR4.3: Structured Logging (ELK)
    - FR4.4: Distributed Tracing
    - FR4.5: Alerting
    - FR4.6: Health Checks
- 4.4 Technical Architecture
    - Monitoring Stack
    - Service Updates
    - Docker Compose
- 4.5 Verification Steps (8 tests)

Phase 5: API Gateway Enhancement

- 5.1 Phase Goals
- 5.2 User Stories (4 stories)
- 5.3 Functional Requirements
    - FR5.1: API Key Management
    - FR5.2: Kong Plugin Configuration
    - FR5.3: API Versioning
    - FR5.4: OpenAPI Documentation
    - FR5.5: Request/Response Logging
    - FR5.6: API Analytics

- 5.4 Technical Architecture
    - Kong Configuration
    - Data Models
- 5.5 Verification Steps (8 tests)

02-PHASES-06-09.md - Core LMS Part 1

Phase 6: Course Management Service

- 6.1 Phase Goals
- 6.2 User Stories (4 stories)
- 6.3 Functional Requirements
    - FR6.1: Course CRUD Operations
    - FR6.2: Subject Management
    - FR6.3: Teacher-Subject Assignment
    - FR6.4: Prerequisites Management
    - FR6.5: Course Catalog API
- 6.4 Technical Architecture
    - New Service Structure
    - Data Models (Course, Subject, TeacherSubjectAssignment)
    - Kong Gateway Update
- 6.5 Verification Steps (6 tests)

Phase 7: Student Enrollment & Allocation

- 7.1 Phase Goals
- 7.2 User Stories (4 stories)
- 7.3 Functional Requirements
    - FR7.1: Enrollment API
    - FR7.2: Section Auto-Allocation
    - FR7.3: Waitlist Management
    - FR7.4: Enrollment Statistics
    - FR7.5: Enrollment Periods
- 7.4 Technical Architecture
    - Data Models (Enrollment, Waitlist, EnrollmentPeriod)
    - Allocation Algorithm
- 7.5 Verification Steps (5 tests)

Phase 8: Content Management System (CMS)

- 8.1 Phase Goals
- 8.2 User Stories (4 stories)
- 8.3 Functional Requirements
    - FR8.1: File Upload
    - FR8.2: Content Access
    - FR8.3: Content Metadata
    - FR8.4: Folder/Module Organization
    - FR8.5: Content Versioning
- 8.4 Technical Architecture
    - New Service Structure
    - Data Models (Content, Folder)
    - S3 Configuration
- 8.5 Verification Steps (6 tests)

Phase 9: Attendance Service

- 9.1 Phase Goals
- 9.2 User Stories (5 stories)
- 9.3 Functional Requirements
    - FR9.1: Attendance Marking
    - FR9.2: QR Code Attendance
    - FR9.3: Attendance Analytics
    - FR9.4: Attendance Reports
    - FR9.5: Regularization Workflow
    - FR9.6: Timetable Integration
- 9.4 Technical Architecture
    - New Service Structure
    - Data Models (AttendanceSession, Attendance, AttendanceRegularization)
- 9.5 Verification Steps (8 tests)

03-PHASES-10-12.md - Core LMS Part 2

Phase 10: Assignment & Submission Service

- 10.1 Phase Goals

- 10.2 User Stories (5 stories)
- 10.3 Functional Requirements
    - FR10.1: Assignment CRUD
    - FR10.2: Submission Workflow
    - FR10.3: Deadline Enforcement
    - FR10.4: Late Submission Handling
    - FR10.5: Plagiarism Detection (Basic)
    - FR10.6: Assignment Types
    - FR10.7: Bulk Operations
- 10.4 Technical Architecture
    - New Service Structure
    - Data Models (Assignment, Submission, PlagiarismCheck)
    - S3 Bucket Structure
- 10.5 Verification Steps (10 tests)

Phase 11: Grading & Assessment Engine

- 11.1 Phase Goals
- 11.2 User Stories (5 stories)
- 11.3 Functional Requirements
    - FR11.1: Manual Grading
    - FR11.2: Rubric-Based Grading
    - FR11.3: Auto-Grading (MCQ)
    - FR11.4: Grade Calculation
    - FR11.5: Grade Analytics
    - FR11.6: Grade Release & Notifications
- 11.4 Technical Architecture
    - Service Updates
    - Data Models (Grade, Rubric, MCQTest, MCQAttempt)
- 11.5 Verification Steps (8 tests)

Phase 12: Real-time Chat Service

- 12.1 Phase Goals
- 12.2 User Stories (5 stories)
- 12.3 Functional Requirements
    - FR12.1: Real-time Messaging (Socket.IO)
    - FR12.2: Chat Rooms

        - FR12.3: Message Persistence (MongoDB)
        - FR12.4: File Sharing
        - FR12.5: Typing Indicators
        - FR12.6: Read Receipts
        - FR12.7: Message Search
        - FR12.8: Notifications
    - 12.4 Technical Architecture
        - New Service Structure
        - MongoDB Schemas
        - Socket.IO Setup
    - 12.5 Verification Steps (10 tests)

04-PHASES-13-18.md - Advanced Features

Phase 13: Search Service (Elasticsearch)

- 13.1 Phase Goals
- 13.2 User Stories (3 stories)
- 13.3 Functional Requirements
    - FR13.1: Elasticsearch Setup
    - FR13.2: Indexing Pipeline
    - FR13.3: Search API
    - FR13.4: Search Analytics
- 13.4 Technical Architecture
- 13.5 Verification Steps (5 tests)

Phase 14: Analytics & Reporting Service

- 14.1 Phase Goals
- 14.2 User Stories (3 stories)
- 14.3 Functional Requirements
    - FR14.1: Student Analytics
    - FR14.2: Teacher Analytics
    - FR14.3: Admin Analytics
    - FR14.4: Pre-built Reports
    - FR14.5: Custom Report Builder
    - FR14.6: Export & Scheduling

- 14.4 Technical Architecture
- 14.5 Verification Steps (5 tests)

Phase 15: Notification Enhancement

- 15.1 Phase Goals
- 15.2 User Stories (3 stories)
- 15.3 Functional Requirements
    - FR15.1: SMS Integration (Twilio)
    - FR15.2: Web Push Notifications
    - FR15.3: Mobile Push (FCM)
    - FR15.4: Notification Preferences
    - FR15.5: Notification History
- 15.4 Technical Architecture
    - Data Models
- 15.5 Verification Steps (5 tests)

Phase 16: Timetable & Scheduling Service

- 16.1 Phase Goals
- 16.2 User Stories (3 stories)
- 16.3 Functional Requirements
    - FR16.1: Timetable CRUD
    - FR16.2: Conflict Detection
    - FR16.3: Room Booking
    - FR16.4: Calendar Integration
    - FR16.5: Recurring Events
- 16.4 Technical Architecture
    - Data Models (Timetable, TimetableSlot, Room, RoomBooking)
- 16.5 Verification Steps (5 tests)

Phase 17: Library Management Integration

- 17.1 Phase Goals
- 17.2 User Stories (3 stories)
- 17.3 Functional Requirements
    - FR17.1: Book Catalog
    - FR17.2: Issue/Return Workflow
    - FR17.3: Fine Calculation

    - FR17.4: Reservation System
    - FR17.5: Integration (Optional)
- 17.4 Technical Architecture
    - Data Models (Book, BookTransaction, BookReservation)
- 17.5 Verification Steps (5 tests)

Phase 18: Payment & Fee Management

- 18.1 Phase Goals
- 18.2 User Stories (3 stories)
- 18.3 Functional Requirements
    - FR18.1: Fee Structure
    - FR18.2: Payment Gateway Integration
    - FR18.3: Receipt Generation
    - FR18.4: Payment History
    - FR18.5: Refund Workflow
- 18.4 Technical Architecture
    - Data Models (FeeStructure, Payment, Refund)
- 18.5 Verification Steps (5 tests)

05-PHASES-19-20.md - Frontend Portals

Phase 19: Super Admin Portal (Next.js)

- 19.1 Phase Goals
- 19.2 User Stories (5 stories)
- 19.3 Functional Requirements
    - FR19.1: Dashboard
    - FR19.2: College Management
    - FR19.3: User Management
    - FR19.4: System Settings
    - FR19.5: Analytics & Reports
    - FR19.6: Audit Logs
- 19.4 Technical Architecture
    - Complete Next.js Project Structure
    - API Integration Examples
    - Authentication Flow

    - Component Examples
- 19.5 Dependencies & Tech Stack
- 19.6 Verification Steps (10 tests)

Phase 20: College Management Portal (Next.js)

- 20.1 Phase Goals
- 20.2 User Stories (6 stories)
- 20.3 Functional Requirements
    - FR20.1: College Dashboard
    - FR20.2: Department & Branch Management
    - FR20.3: Course Management
    - FR20.4: Faculty Management
    - FR20.5: Student Management
    - FR20.6: Timetable Management
    - FR20.7: Attendance Management
    - FR20.8: Reports & Analytics
    - FR20.9: Announcements & Notifications
    - FR20.10: Settings
- 20.4 Technical Architecture
    - Complete Next.js Project Structure
    - Key Component Examples (BulkImport, TimetableGrid)
- 20.5 Verification Steps (10 tests)

06-PHASES-21-25.md - Production & AI/ML

Phase 21: Student LMS Portal (Next.js)

- 21.1 Phase Goals
- 21.2 User Stories (6 stories)
- 21.3 Functional Requirements
    - FR21.1: Student Dashboard
    - FR21.2: Course Catalog & Enrollment
    - FR21.3: My Courses
    - FR21.4: Assignments
    - FR21.5: Grades
    - FR21.6: Course Content

        - FR21.7: Chat
        - FR21.8: Attendance
        - FR21.9: Timetable
        - FR21.10: Profile & Settings
    - 21.4 Technical Architecture
        - Complete Project Structure
        - Socket.IO Integration
        - Service Worker (Offline Support)
    - 21.5 Verification Steps (10 tests)

Phase 22: AWS Infrastructure (Terraform)

- 22.1 Phase Goals
- 22.2 Infrastructure Components
    - IC22.1: Networking (VPC)
    - IC22.2: Compute (EKS)
    - IC22.3: Database (RDS)
    - IC22.4: Messaging (MSK)
    - IC22.5: Caching (Elasticache)
    - IC22.6: Storage (S3)
    - IC22.7: Monitoring & Logging
    - IC22.8: Security
    - IC22.9: CI/CD Integration
- 22.3 Terraform Structure
    - Modules
    - Environments
    - Kubernetes Manifests
- 22.4 Cost Estimation
- 22.5 Verification Steps (10 tests)

Phase 23: CI/CD Pipeline (GitHub Actions)

- 23.1 Phase Goals
- 23.2 CI/CD Workflows
    - Workflow 1: Build & Test
    - Workflow 2: Build Docker Image
    - Workflow 3: Deploy to Dev
    - Workflow 4: Deploy to Staging

    - Workflow 5: Deploy to Production
    - Workflow 6: Rollback
    - Workflow 7: Database Migrations
- 23.3 GitHub Actions Structure
- 23.4 Example Workflow (Production Deployment)
- 23.5 Verification Steps (10 tests)

Phase 24: Mobile Applications (React Native)

- 24.1 Phase Goals
- 24.2 Mobile App Features (10 features)
- 24.3 Technical Architecture
    - Complete React Native Project Structure
    - Key Implementation Examples
- 24.4 App Store Deployment
    - iOS (App Store)
    - Android (Play Store)
- 24.5 Verification Steps (10 tests)

Phase 25: AI/ML Features

- 25.1 Phase Goals
- 25.2 AI/ML Features
    - ML25.1: Course Recommendation Engine
    - ML25.2: AI-Powered Essay Grading
    - ML25.3: Online Exam Proctoring
    - ML25.4: AI Chatbot
    - ML25.5: Predictive Analytics (At-Risk Students)
- 25.3 Technical Architecture
    - AI Service Structure
    - Python Implementation Examples
- 25.4 Model Training Pipeline
- 25.5 Verification Steps (8 tests)

07-APPENDIX.md - Reference Materials

A. Complete Data Models

- A.1 Authentication & User Management
- A.2 Academic Structure
- A.3 Course Management
- A.4 Content & Assignments
- A.5 Attendance & Grading
- A.6 Communication
- A.7 Analytics & Reporting
- A.8 System Configuration

B. API Contracts

- B.1 Authentication APIs
- B.2 Course APIs
- B.3 Assignment APIs
- B.4 Content APIs
- B.5 Chat APIs
- B.6 Analytics APIs

C. Kafka Topics Reference

- C.1 Authentication Events
- C.2 Course Events
- C.3 Assignment Events
- C.4 Notification Events
- C.5 Analytics Events

D. Environment Variables

- D.1 Auth Service
- D.2 Course Service
- D.3 Content Service
- D.4 Chat Service
- D.5 Infrastructure

E. Glossary

- E.1 Technical Terms
- E.2 Academic Terms
- E.3 Acronyms

F. Migration Guide

- F.1 From Existing LMS
- F.2 Data Migration Steps
- F.3 User Migration

G. Security Best Practices

- G.1 Authentication Security
- G.2 API Security
- G.3 Data Protection
- G.4 Infrastructure Security

H. Performance Benchmarks

- H.1 API Response Times
- H.2 Database Query Performance
- H.3 Frontend Load Times
- H.4 Scalability Metrics

PHASE-BREAKDOWN.md - Quick Reference

Complete Phase List (1-25)

- Foundation & Security (Phases 1-5)
- Core LMS Features (Phases 6-12)
- Advanced Features (Phases 13-18)
- Frontend Applications (Phases 19-21)
- Production & Scale (Phases 22-25)

Implementation Timeline

- Year 1: Phases 1-11 (MVP & Core Features)
- Year 2: Phases 12-21 (Advanced Features & Frontend)
- Year 3: Phases 22-25 (Production & Scale)

README.md - Navigation Guide

How to Use This PRD

- For Product Managers
- For Engineers
- For QA/Testing
- For DevOps

Development Workflow

- Phase Execution Process
- Typical Phase Duration

Contact & Support

- Team Contacts
- Slack Channels
- Documentation Links

Quick Navigation by Topic

Authentication & Security

- Phase 1: Auth Service Hardening → 01-PHASES-01-05.md
- Phase 2: Single Device Login → 01-PHASES-01-05.md
- Phase 5: API Gateway Enhancement → 01-PHASES-01-05.md
- Appendix G: Security Best Practices → 07-APPENDIX.md

Course Management

- Phase 6: Course Management Service → 02-PHASES-06-09.md
- Phase 7: Student Enrollment → 02-PHASES-06-09.md
- Phase 8: Content Management → 02-PHASES-06-09.md

Assignments & Grading

- Phase 10: Assignment & Submission → 03-PHASES-10-12.md
- Phase 11: Grading & Assessment → 03-PHASES-10-12.md

Communication

- Phase 12: Real-time Chat → 03-PHASES-10-12.md
- Phase 15: Notification Enhancement → 04-PHASES-13-18.md

Analytics & Reporting

- Phase 13: Search Service → 04-PHASES-13-18.md
- Phase 14: Analytics & Reporting → 04-PHASES-13-18.md

Frontend Applications

- Phase 19: Super Admin Portal → 05-PHASES-19-20.md
- Phase 20: College Management Portal → 05-PHASES-19-20.md
- Phase 21: Student LMS Portal → 06-PHASES-21-25.md

Infrastructure & DevOps

- Phase 4: Monitoring & Observability → 01-PHASES-01-05.md
- Phase 22: AWS Infrastructure → 06-PHASES-21-25.md
- Phase 23: CI/CD Pipeline → 06-PHASES-21-25.md

Advanced Features

- Phase 16: Timetable & Scheduling → 04-PHASES-13-18.md

- Phase 17: Library Management → 04-PHASES-13-18.md
- Phase 18: Payment & Fee Management → 04-PHASES-13-18.md
- Phase 24: Mobile Applications → 06-PHASES-21-25.md
- Phase 25: AI/ML Features → 06-PHASES-21-25.md

Document Statistics

Total Content

- **Documents**: 10 files
- **Phases**: 25 detailed phases
- **User Stories**: 100+ stories
- **Functional Requirements**: 200+ requirements
- **API Endpoints**: 150+ endpoints
- **Data Models**: 50+ models
- **Verification Steps**: 200+ test cases
- **Code Examples**: 50+ examples
- **Diagrams**: 25+ architecture diagrams (text-based)

Word Count by Document

Document	Approximate Words
00-OVERVIEW.md	3,500
01-PHASES-01-05.md	12,000
02-PHASES-06-09.md	10,000
03-PHASES-10-12.md	9,000
04-PHASES-13-18.md	7,500
05-PHASES-19-20.md	10,000
06-PHASES-21-25.md	12,000
07-APPENDIX.md	6,000

Document	Approximate Words
PHASE-BREAKDOWN.md	2,500
README.md	1,500
Total	**~74,000 words**

Revision History

Version	Date	Author	Changes
1.0	December 2025	Product & Engineering Team	Initial comprehensive PRD
1.1	TBD	TBD	Post-stakeholder review updates

Document Maintenance

Review Cycle: Quarterly or after each phase completion

Update Process:

1. Gather feedback from implementation team
2. Update relevant sections
3. Increment version number
4. Notify stakeholders
5. Archive previous version

Ownership:

- **Product Manager**: Overall PRD ownership
- **Technical Lead**: Architecture and technical accuracy
- **Project Manager**: Timeline and resource planning

Last Updated: December 15, 2025
Next Review: March 2026

LMS Platform PRD - Phases 1-5

Foundation & Security Enhancement

Phase 1: Auth Service Hardening & Refinement

Phase Goals

1. Complete all pending auth-service improvements identified in EXISTING_COMPONENTS_IMPROVEMENTS.md
2. Implement comprehensive error handling and logging across all auth endpoints
3. Add rate limiting and brute-force protection
4. Enhance token refresh mechanism with rotation
5. Implement password policy enforcement

User Stories

ID	User Story	Priority	Acceptance Criteria
US1.1	As a **System Administrator**, I want comprehensive audit logs for all authentication events so that I can track security incidents and comply with regulations	Must-Have	- All login/logout/register events logged to Kafka topic `auth.audit` - Logs include IP, user agent, timestamp, outcome - Logs retained for 90 days

ID	User Story	Priority	Acceptance Criteria
			- Searchable via Elasticsearch (future)
US1.2	As a **Student**, I want my account locked after 5 failed login attempts so that my account is protected from brute-force attacks	Must-Have	- Account locked for 15 minutes after 5 failures - Email notification sent - Admin can manually unlock - Counter resets on successful login
US1.3	As a **Teacher**, I want to set a strong password that meets security requirements so that my account is secure	Must-Have	- Minimum 12 characters - Requires uppercase, lowercase, number, special char - Cannot reuse last 5 passwords - Password strength meter in UI (future)
US1.4	As a **College Admin**, I want API endpoints to respond with consistent error formats so that frontend integration is predictable	Should-Have	- All errors follow RFC 7807 Problem Details - Include error code, message, trace ID - Validation errors list all fields - HTTP status codes match semantics

Functional Requirements

FR1.1: Audit Logging

- Log all authentication events (login, logout, register, password reset, email verify) to Kafka topic `auth.audit.events`
- Event schema: `{ userId, userType, action, ipAddress, userAgent, timestamp, success, errorCode?, metadata }`
- Implement log correlation IDs for request tracing
- Sensitive data (passwords, tokens) must be excluded from logs

FR1.2: Account Lockout

- Track failed login attempts in Redis: `auth:failed:{userType}:{userId}` with TTL 15 minutes
- Increment counter on each failure
- Lock account (set `isLocked: true` in DB + Redis flag) after 5 attempts
- Send email notification via Kafka `notification.account.locked` topic
- Provide admin endpoint `POST /api/v1/auth/admin/unlock-account` (requires ADMIN role)
- Auto-unlock after 15 minutes (background job or TTL-based)

FR1.3: Password Policy

- Validate password on registration and password change:
  - Length: 12-128 characters
  - Complexity: At least 1 uppercase, 1 lowercase, 1 digit, 1 special character
  - No common passwords (check against top 10k list)
  - No sequential characters (e.g., “123456”, “abcdef”)
- Store password history (hashed) in `PasswordHistory` table (last 5 passwords)
- Reject password if matches any in history
- Implement password expiry (90 days) with grace period (7 days warning)

FR1.4: Token Refresh Enhancement

- Implement refresh token rotation: issue new refresh token on each use
- Invalidate old refresh token immediately
- Store refresh token family ID to detect token reuse (security breach indicator)
- If reused token detected, invalidate entire token family and force re-login
- Refresh tokens expire after 30 days of inactivity

FR1.5: Rate Limiting

- Implement rate limiting middleware using Redis:
  - Login endpoints: 5 requests/minute per IP
  - Registration: 3 requests/hour per IP
  - Password reset: 3 requests/hour per user
  - OTP verification: 10 requests/hour per user
- Return HTTP 429 with `Retry-After` header
- Whitelist internal service IPs

FR1.6: Error Standardization

- Implement RFC 7807 Problem Details format:

```
{
  "type": "https://lms.example.com/errors/invalid-credentials",
  "title": "Invalid Credentials",
  "status": 401,
  "detail": "The email or password provided is incorrect",
  "instance": "/api/v1/auth/student/login",
  "traceId": "abc-123-def",
  "timestamp": "2025-12-15T20:50:00Z"
}
```

- Validation errors include field-level details:

```
{
  "type": "https://lms.example.com/errors/validation-failed",
  "title": "Validation Failed",
  "status": 400,
  "errors": [
    { "field": "email", "message": "Invalid email format" },
    { "field": "password", "message": "Password too short" }
  ]
}
```

Non-Functional Requirements

NFR1.1: Performance

- Login endpoint response time: <300ms (p95)
- Token refresh: <100ms (p95)
- Rate limiting check overhead: <10ms

NFR1.2: Security

- All passwords hashed with Argon2id (time cost: 3, memory: 64MB, parallelism: 4)
- Refresh tokens stored hashed in database
- Audit logs encrypted at rest
- TLS 1.3 for all connections

NFR1.3: Reliability

- Redis failover handling (graceful degradation if Redis unavailable)
- Kafka producer retries (3 attempts with exponential backoff)
- Database connection pooling (min: 5, max: 20)

NFR1.4: Observability

- Prometheus metrics: `auth_login_attempts_total`, `auth_lockouts_total`, `auth_token_refresh_total`
- Structured JSON logging (Winston/Pino)
- Distributed tracing with correlation IDs

Technical Architecture

Updated Components:

```
services/auth-service/
├── src/
│   ├── middleware/
│   │   ├── rateLimiter.ts        [NEW] Redis-based rate limiting
│   │   ├── auditLogger.ts        [NEW] Kafka audit event producer
│   │   └── authValidator.ts      [UPDATED] Add lockout check
│   ├── services/
```

```
│   │   ├── password.service.ts     [NEW] Password validation, history
│   │   ├── lockout.service.ts      [NEW] Account lockout logic
│   │   └── token.service.ts        [UPDATED] Refresh token rotation
│   ├── utils/
│   │   ├── errorFormatter.ts       [NEW] RFC 7807 formatter
│   │   └── passwordValidator.ts    [NEW] Password policy rules
│   ├── config/
│   │   └── rateLimits.ts           [NEW] Rate limit configurations
│   └── types/
│       └── audit.types.ts          [NEW] Audit event schemas
├── prisma/
│   └── schema.prisma               [UPDATED] Add PasswordHistory,
AccountLockout models
└── tests/
    ├── unit/
    │   ├── password.service.test.ts [NEW]
    │   └── lockout.service.test.ts  [NEW]
    └── integration/
        └── auth-lockout.test.ts     [NEW]
```

Data Model Updates (Prisma):

```
model PasswordHistory {
  id        String   @id @default(uuid())
  userId    String
  userType  String   // "student", "teacher", etc.
  passwordHash String
  createdAt DateTime @default(now())

  @@index([userId, userType])
  @@index([createdAt])
}

model AccountLockout {
```

```
  id            String   @id @default(uuid())
  userId        String   @unique
  userType      String
  lockedAt      DateTime @default(now())
  lockedUntil   DateTime
  reason        String   // "brute_force", "admin_action"
  unlockToken   String?  // For admin unlock

  @@index([userId, userType])
}

// Update existing User models (Student, Teacher, etc.)
model Student {
  // ... existing fields
  passwordChangedAt DateTime?
  passwordExpiresAt DateTime?
  isLocked          Boolean  @default(false)
  lockReason        String?
}
```

Kafka Topics:

- `auth.audit.events` (new): Audit logs
- `notification.account.locked` (new): Account lockout notifications

Redis Keys:

- `auth:failed:{userType}:{userId}`: Failed login counter (TTL: 15 min)
- `auth:locked:{userType}:{userId}`: Lock flag (TTL: 15 min or manual unlock)
- `ratelimit:{endpoint}:{ip}`: Rate limit counters (TTL: varies)
- `token:family:{familyId}`: Refresh token family tracking

Dependencies & Tech Stack Updates

New Dependencies:

- `argon2`: Password hashing (replace bcrypt if used)

- `ioredis`: Redis client (if not already present)
- `express-rate-limit`: Rate limiting middleware
- `rate-limit-redis`: Redis store for rate limiter
- `zxcvbn`: Password strength estimation
- `validator`: Email/input validation

Configuration:

- Environment variables: `LOCKOUT_THRESHOLD`, `LOCKOUT_DURATION`, `PASSWORD_EXPIRY_DAYS`

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V1.1	Failed login lockout	Postman/Jest	- 5 failed logins lock account - 6th attempt returns 423 Locked - Email notification sent - Redis key exists with correct TTL
V1.2	Password policy enforcement	Jest	- Weak passwords rejected with specific errors - Password history prevents reuse - Strong passwords accepted
V1.3	Token refresh rotation	Postman	- Refresh returns new access + refresh tokens - Old refresh token invalidated - Reused token triggers security alert
V1.4	Rate limiting	Artillery	- 6 login requests/min from same IP returns 429 - Different IPs not affected - Retry-After header present
V1.5	Audit logging	Kafka UI	- All auth events in `auth.audit.events` topic - Events have correct schema - No sensitive data in logs

ID	Test Case	Tool	Acceptance Criteria
V1.6	Error format consistency	Postman	- All errors follow RFC 7807 - Validation errors list all fields - Trace IDs present and unique
V1.7	Admin unlock account	Postman	- Admin can unlock locked account - User can login after unlock - Unlock event logged
V1.8	Password expiry	Manual/ Cron	- Users warned 7 days before expiry - Login blocked after expiry - Password change resets expiry

Load Testing:

- Artillery scenario: 100 concurrent users, 1000 login attempts over 60 seconds
- Expected: <1% failure rate, p95 response time <500ms, no lockouts for valid credentials

Security Testing:

- OWASP ZAP scan for auth endpoints
- Brute-force simulation (Hydra/Burp Suite)
- Token reuse detection test

Phase 2: Single Device Login Enforcement

Phase Goals

1. Complete single device login enforcement across all user types
2. Implement real-time session invalidation using Redis Pub/Sub
3. Add device fingerprinting and management UI (backend)
4. Handle race conditions for simultaneous logins
5. Provide user notifications for new device logins

User Stories

ID	User Story	Priority	Acceptance Criteria
US2.1	As a **Student**, I want to be automatically logged out from my old device when I login from a new device so that my account security is maintained	Must-Have	- Login on Device B logs out Device A within 5 seconds - Device A shows “Session expired” message - Email notification sent about new device login
US2.2	As a **Teacher**, I want to see which device is currently logged in with my account so that I can detect unauthorized access	Should-Have	- API endpoint returns current device info (type, OS, browser, IP, login time) - Device fingerprint stored securely
US2.3	As a **College Admin**, I want to remotely logout a user from all devices so that I can respond to security incidents	Must-Have	- Admin endpoint `/api/v1/auth/admin/force-logout/{userId}` works - All active sessions invalidated - User must re-login - Action logged in audit trail
US2.4	As a **HOD**, I want to receive a notification when my account is accessed from a new device so that I can detect suspicious activity	Should-Have	- Email sent immediately on new device login - Notification includes device details, location (IP-based), timestamp - Option to “Secure

ID	User Story	Priority	Acceptance Criteria
			Account" (logout all devices)

Functional Requirements

FR2.1: Device Fingerprinting

- Generate device fingerprint from:
  - User-Agent (browser, OS, device type)
  - IP address (hashed for privacy)
  - Accept-Language header
  - Screen resolution (from client, optional)
- Hash fingerprint using SHA-256: `deviceId = SHA256(userAgent + ipHash + language)`
- Store in database: `UserSession` table

FR2.2: Session Management

- On login, check if active session exists for user:
  - If yes, invalidate old session (delete from DB + Redis)
  - Publish to Redis Pub/Sub: `session:invalidate:{oldSessionId}`
  - Create new session with new `deviceId`
- Store session in:
  - **Database (PostgreSQL)**: `UserSession` table (persistent record)
  - **Redis**: `session:{sessionId}` (fast lookup, TTL = token expiry)

FR2.3: Real-time Invalidation

- Implement Redis Pub/Sub for session invalidation:
  - Publisher: Auth service on new login
  - Subscriber: All auth-service instances + Kong Gateway (via Lua script update)
- On receiving `session:invalidate:{sessionId}`:
  - Delete from Redis cache
  - Mark session as `invalidated` in DB
  - WebSocket notification to client (if connected)

FR2.4: Race Condition Handling

- Use Redis distributed lock for login process:

```
LOCK: auth:login:{userType}:{userId} (TTL: 5 seconds)
1. Acquire lock
2. Check existing session
3. Invalidate old session
4. Create new session
5. Release lock
```

- If lock acquisition fails, retry up to 3 times with 100ms delay

FR2.5: Device Management API

- `GET /api/v1/auth/me/session`: Get current session details
  - Response: `{ sessionId, deviceId, deviceType, browser, os, ipAddress, loginAt, expiresAt }`
- `POST /api/v1/auth/me/logout-all`: Logout from all devices (user-initiated)
- `POST /api/v1/auth/admin/force-logout`: Admin force logout (requires ADMIN role)
  - Body: `{ userId, userType, reason }`

FR2.6: New Device Notifications

- On new device login, publish to Kafka: `notification.new.device.login`
- Notification payload:

```
{
  "userId": "...",
  "userType": "student",
  "deviceType": "Mobile",
  "browser": "Chrome 120",
  "os": "Android 14",
  "ipAddress": "192.168.1.1",
  "location": "Mumbai, India",
  "loginAt": "2025-12-15T20:50:00Z",
```

```
  "secureAccountUrl": "https://lms.example.com/secure-account?
                token=..."
}
```

- Email template includes:
  - Device details
  - Login time and location
  - “Was this you?” prompt
  - One-click “Secure Account” button (logout all devices)

Non-Functional Requirements

NFR2.1: Performance

- Session lookup from Redis: <10ms
- Login with session invalidation: <500ms total
- Redis Pub/Sub message delivery: <100ms

NFR2.2: Consistency

- Eventual consistency acceptable (5-second window for invalidation propagation)
- Database as source of truth for session state

NFR2.3: Scalability

- Support 10,000 concurrent sessions
- Redis Pub/Sub handles 1000 messages/second

NFR2.4: Reliability

- If Redis Pub/Sub fails, fallback to polling (check DB every 30 seconds)
- Session data persisted in DB (survives Redis restart)

Technical Architecture

Updated Components:

```
services/auth-service/
├── src/
```

```
│   ├── services/
│   │   ├── session.service.ts      [NEW] Session CRUD, invalidation
│   │   ├── device.service.ts       [NEW] Device fingerprinting
│   │   └── pubsub.service.ts       [NEW] Redis Pub/Sub wrapper
│   ├── middleware/
│   │   └── sessionValidator.ts     [NEW] Check session validity on each
request
│   ├── controllers/
│   │   └── session.controller.ts   [NEW] Session management endpoints
│   ├── utils/
│   │   ├── deviceFingerprint.ts    [NEW] Generate device ID
│   │   └── distributedLock.ts      [NEW] Redis lock implementation
│   └── types/
│       └── session.types.ts        [NEW] Session interfaces
├── prisma/
│   └── schema.prisma               [UPDATED] Add UserSession model
└── tests/
    ├── integration/
    │   ├── single-device-login.test.ts [NEW]
    │   └── race-condition.test.ts      [NEW]
    └── load/
        └── concurrent-logins.artillery.yml [NEW]
```

Data Model (Prisma):

```
model UserSession {
  id           String   @id @default(uuid())
  userId       String
  userType     String   // "student", "teacher", etc.
  sessionToken String   @unique // PASETO token ID
  deviceId     String   // SHA-256 hash of device fingerprint
  deviceType   String?  // "Mobile", "Desktop", "Tablet"
  browser      String?
  os           String?
```

```
  ipAddress     String   // Hashed for privacy
  location      String?  // IP geolocation (city, country)
  createdAt     DateTime @default(now())
  expiresAt     DateTime
  invalidatedAt DateTime?
  isActive      Boolean  @default(true)

  @@index([userId, userType, isActive])
  @@index([sessionToken])
  @@index([expiresAt])
}
```

Redis Data Structures:

- `session:{sessionId}`: Hash with session details (TTL = token expiry)

```
{
  userId, userType, deviceId, createdAt, expiresAt
}
```

- `user:active:session:{userType}:{userId}`: String with current sessionId (TTL = token expiry)
- Pub/Sub channel: `session:invalidate` (messages: `{sessionId}`)

Kong Gateway Update:

- Update `paseto-vault-auth-lua/handler.lua` to check Redis for session validity:

```lua
local sessionId = token.payload.sid
local session = redis:get("session:" .. sessionId)
if not session then
  return kong.response.exit(401, {error = "Session invalidated"})
end
```

Notification Service Update:

```
services/notification-service/
├── src/
│   ├── handlers/
│   │   └── new-device-login.handler.ts [NEW]
│   └── templates/
│       └── new-device-login.template.ts [NEW]
```

Dependencies & Tech Stack Updates

New Dependencies:

- `ua-parser-js`: User-Agent parsing
- `geoip-lite`: IP geolocation (or AWS Location Service)
- `redlock`: Distributed locking (or custom implementation)

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V2.1	Single device enforcement	Postman	- Login on Device B invalidates Device A session - Device A's next API call returns 401 - Only 1 active session in DB
V2.2	Real-time invalidation	Manual (2 browsers)	- Device A logged out within 5 seconds of Device B login - Redis Pub/Sub message received - WebSocket notification shown (if implemented)
V2.3	Race condition handling	Jest (concurrent requests)	- 10 simultaneous logins for same user - Only 1 session active at end - No duplicate sessions in DB - All logins succeed (last one wins)

ID	Test Case	Tool	Acceptance Criteria
V2.4	Device fingerprinting	Postman	- Different browsers generate different deviceIds - Same browser generates same deviceId - deviceId is SHA-256 hash (64 chars)
V2.5	New device notification	Kafka UI + Email	- Email sent on new device login - Contains correct device details - “Secure Account” link works
V2.6	Admin force logout	Postman	- Admin can logout any user - All user’s sessions invalidated - User’s next request returns 401 - Action logged in audit trail
V2.7	Session API endpoints	Postman	- GET /me/session returns current session - POST /me/logout-all invalidates all sessions - User must re-login after logout-all
V2.8	Kong Gateway integration	Postman	- Kong rejects requests with invalidated session - Returns 401 with clear error message

Load Testing:

- Artillery: 500 users login simultaneously (same user, different devices)
- Expected: Last login succeeds, all others invalidated, no race conditions

Chaos Testing:

- Redis Pub/Sub failure simulation (stop Redis)
- Expected: Fallback to DB polling, eventual consistency maintained

---

Phase 3: Bulk Import Infrastructure

Phase Goals

1. Implement asynchronous bulk import for all user types (students, teachers, staff)
2. Build Kafka-based job queue with worker service
3. Add progress tracking and error reporting
4. Support CSV and JSON formats with validation
5. Implement rollback mechanism for failed imports

User Stories

ID	User Story	Priority	Acceptance Criteria
US3.1	As a **College Admin**, I want to upload a CSV file with 5000 student records and have them imported in the background so that I don't have to wait for the process to complete	Must-Have	- CSV upload returns job ID immediately - Import happens asynchronously - Progress visible via API - Completion notification sent
US3.2	As a **HOD**, I want to see detailed error reports for failed bulk imports so that I can fix data issues and retry	Must-Have	- Error report downloadable as CSV - Shows row number, field, error message - Partial success supported (valid rows imported) - Invalid rows

ID	User Story	Priority	Acceptance Criteria
			skipped with report
US3.3	As a **Dean**, I want to validate my bulk import file before submitting so that I can catch errors early	Should-Have	- Validation-only endpoint available - Returns errors without importing - Fast validation (<10 seconds for 5000 rows)
US3.4	As a **College Admin**, I want to rollback a bulk import if I uploaded wrong data so that I can correct mistakes	Could-Have	- Rollback endpoint available within 24 hours of import - All imported records deleted - Rollback action logged

Functional Requirements

FR3.1: File Upload & Validation

- Endpoints:
  - `POST /api/v1/bulk/students/validate`: Validate CSV/JSON without importing
  - `POST /api/v1/bulk/students/import`: Upload and import
  - Similar endpoints for teachers, staff, courses, etc.
- File upload:
  - Max file size: 50 MB
  - Supported formats: CSV, JSON
  - Upload to S3 bucket: `lms-bulk-imports/{jobId}/{filename}`

- Validation rules:
  - Required fields present
  - Data types correct (email format, phone format, dates)
  - Referential integrity (e.g., batchId exists)
  - Duplicate detection (within file and against DB)
- Validation response:

```
{
  "valid": false,
  "totalRows": 5000,
  "validRows": 4850,
  "invalidRows": 150,
  "errors": [
    { "row": 23, "field": "email", "message": "Invalid email format" },
    { "row": 45, "field": "batchId", "message": "Batch not found" }
  ]
}
```

FR3.2: Asynchronous Job Processing

- On import request:
  1. Upload file to S3
  2. Create job record in DB (`BulkImportJob` table)
  3. Publish to Kafka topic: `bulk.import.jobs`
  4. Return job ID to client
- Job payload:

```
{
  "jobId": "uuid",
  "type": "student_import",
  "fileUrl": "s3://...",
  "uploadedBy": "userId",
  "collegeId": "...",
  "options": { "skipDuplicates": true, "sendWelcomeEmails": true }
}
```

FR3.3: Worker Service

- New microservice: `bulk-import-service`
- Consumes from `bulk.import.jobs` topic
- Processing logic:
    1. Download file from S3
    2. Parse CSV/JSON (streaming for large files)
    3. Validate each row
    4. Insert valid rows in batches (100 rows/batch)
    5. Update job progress in DB and Redis
    6. Generate error report for invalid rows
    7. Upload error report to S3
    8. Publish completion event to Kafka: `bulk.import.completed`
- Idempotency: Use job ID to prevent duplicate processing

FR3.4: Progress Tracking

- Store progress in Redis: `bulk:job:{jobId}:progress`

```
{
  "jobId": "...",
  "status": "processing", // "pending", "processing", "completed", "failed"
  "totalRows": 5000,
  "processedRows": 2500,
  "successRows": 2450,
  "failedRows": 50,
  "startedAt": "...",
  "estimatedCompletion": "..."
}
```

- API endpoint: `GET /api/v1/bulk/jobs/{jobId}/status`
- WebSocket support for real-time updates (optional)

FR3.5: Error Reporting

- Generate error CSV with columns: `Row Number, Field, Value, Error Message`
- Upload to S3: `lms-bulk-imports/{jobId}/errors.csv`
- Include in job completion notification

- API endpoint: `GET /api/v1/bulk/jobs/{jobId}/errors` (returns S3 presigned URL)

FR3.6: Rollback Mechanism

- Store import metadata: `BulkImportRecord` table (jobId, recordId, recordType)
- Rollback endpoint: `POST /api/v1/bulk/jobs/{jobId}/rollback`
- Rollback logic:
  1. Fetch all records imported in this job
  2. Soft delete (set `deletedAt` timestamp) or hard delete
  3. Publish rollback event to Kafka
  4. Update job status to `rolled_back`
- Rollback only allowed within 24 hours and if job status is `completed`

FR3.7: Notification Integration

- On job completion, publish to `notification.bulk.import.completed`:

```
{
  "jobId": "...",
  "userId": "...",
  "type": "student_import",
  "status": "completed",
  "totalRows": 5000,
  "successRows": 4850,
  "failedRows": 150,
  "errorReportUrl": "https://...",
  "completedAt": "..."
}
```

- Email template includes summary and link to error report

Non-Functional Requirements

NFR3.1: Performance

- Import throughput: 1000 rows/minute (single worker)
- Validation: 5000 rows in <10 seconds
- File upload: Support up to 50 MB (streaming upload)

NFR3.2: Scalability

- Horizontal scaling: Multiple worker instances (Kafka consumer group)
- Handle 100 concurrent import jobs

NFR3.3: Reliability

- Worker retries on failure (3 attempts with exponential backoff)
- Dead letter queue for permanently failed jobs
- Job timeout: 1 hour (mark as failed if not completed)

NFR3.4: Data Integrity

- Transactional batch inserts (rollback batch on error)
- Duplicate detection (check existing records before insert)
- Referential integrity checks (foreign keys validated)

Technical Architecture

New Service:

```
services/bulk-import-service/
├── Dockerfile
├── package.json
├── tsconfig.json
├── src/
│   ├── index.ts
│   ├── config/
│   │   ├── env.ts
│   │   ├── s3.ts
│   │   └── kafka.ts
│   ├── workers/
│   │   ├── student-import.worker.ts
│   │   ├── teacher-import.worker.ts
│   │   └── base.worker.ts
│   ├── services/
│   │   ├── parser.service.ts       // CSV/JSON parsing
│   │   ├── validator.service.ts    // Row validation
```

```
│   │   ├── importer.service.ts     // DB insertion
│   │   └── progress.service.ts     // Progress tracking
│   ├── utils/
│   │   ├── csvParser.ts
│   │   ├── jsonParser.ts
│   │   └── errorReporter.ts
│   └── types/
│       ├── job.types.ts
│       └── import.types.ts
├── prisma/
│   └── schema.prisma
└── tests/
    ├── unit/
    │   └── validator.service.test.ts
    └── integration/
        └── student-import.test.ts
```

Auth Service Updates:

```
services/auth-service/
├── src/
│   ├── controllers/
│   │   └── bulk/
│   │       ├── bulk-import.controller.ts [NEW]
│   │       └── bulk-job.controller.ts    [NEW]
│   ├── services/
│   │   └── bulk-upload.service.ts        [NEW] S3 upload, job creation
│   └── routes/
│       └── bulk.route.ts                 [NEW]
```

Data Models:

```
model BulkImportJob {
  id          String   @id @default(uuid())
```

```
  type          String   // "student_import", "teacher_import"
  fileUrl       String   // S3 URL
  fileName      String
  uploadedBy    String
  uploadedByType String
  collegeId     String?
  status        String   // "pending", "processing", "completed", "failed",
"rolled_back"
  totalRows     Int?
  successRows   Int      @default(0)
  failedRows    Int      @default(0)
  errorReportUrl String?
  options       Json?    // { skipDuplicates, sendWelcomeEmails }
  startedAt     DateTime?
  completedAt   DateTime?
  createdAt     DateTime @default(now())

  @@index([status, createdAt])
  @@index([uploadedBy])
}

model BulkImportRecord {
  id         String   @id @default(uuid())
  jobId      String
  recordId   String   // ID of imported record (studentId, teacherId)
  recordType String   // "student", "teacher"
  createdAt  DateTime @default(now())

  @@index([jobId])
  @@index([recordId, recordType])
}
```

Kafka Topics:

- `bulk.import.jobs`: Job requests

- `bulk.import.completed`: Job completion events
- `bulk.import.failed`: Job failure events

S3 Bucket Structure:

```
lms-bulk-imports/
├── {jobId}/
│   ├── original/
│   │   └── students.csv
│   └── errors/
│       └── errors.csv
```

Dependencies & Tech Stack Updates

New Dependencies:

- `csv-parser`: CSV parsing
- `papaparse`: Alternative CSV parser
- `fast-csv`: Fast CSV writing (for error reports)
- `aws-sdk` / `@aws-sdk/client-s3`: S3 integration
- `bull` or `kafkajs`: Job queue (already have Kafka)

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V3.1	CSV upload and validation	Postman	- Upload 5000-row CSV - Validation completes in <10s - Returns detailed error report
V3.2	Async import processing	Postman + Kafka UI	- Import returns job ID immediately - Job appears in Kafka topic - Worker processes job - Progress updates in Redis
V3.3	Progress tracking	Postman (polling)	- GET /jobs/{id}/status returns current progress

ID	Test Case	Tool	Acceptance Criteria
			- Progress updates every 10 seconds - Final status is “completed”
V3.4	Error reporting	Postman	- Failed rows generate error CSV - Error CSV downloadable from S3 - Contains row number, field, error
V3.5	Partial success	Manual	- File with 100 valid + 50 invalid rows - 100 rows imported successfully - 50 rows in error report - Job status “completed”
V3.6	Rollback mechanism	Postman	- Import 100 students - Rollback job - All 100 students deleted/soft-deleted - Job status “rolled_back”
V3.7	Duplicate detection	Manual	- Import same file twice - Second import skips duplicates - No duplicate records in DB
V3.8	Completion notification	Email	- Email sent on job completion - Contains summary (success/failed counts) - Link to error report works

Load Testing:

- Import 10,000 students (single job)
- Expected: Completes in <10 minutes, <1% failure rate
- Concurrent: 10 jobs of 1000 students each
- Expected: All complete within 15 minutes

Stress Testing:

- Upload 50 MB CSV file
- Expected: Upload succeeds, parsing doesn’t crash worker

Phase 4: Monitoring & Observability Foundation

Phase Goals

1. Set up Prometheus for metrics collection
2. Configure Grafana with pre-built dashboards
3. Implement structured logging with Elasticsearch + Kibana (ELK)
4. Add distributed tracing with OpenTelemetry
5. Set up alerting for critical issues

User Stories

ID	User Story	Priority	Acceptance Criteria
US4.1	As a **DevOps Engineer**, I want real-time metrics dashboards so that I can monitor system health and performance	Must-Have	- Grafana dashboard shows CPU, memory, request rate, error rate - Metrics updated every 15 seconds - Historical data retained for 30 days
US4.2	As a **Backend Developer**, I want to search application logs by trace ID so that I can debug issues quickly	Must-Have	- Logs searchable in Kibana - Trace ID links all related logs - Search results in <2 seconds
US4.3	As a **Site Reliability Engineer**, I want to receive alerts when API error rate exceeds 5% so that I can respond to incidents proactively	Must-Have	- Alert sent to Slack/ Email within 1 minute - Alert includes service name, error rate, time

ID	User Story	Priority	Acceptance Criteria
			window - Runbook link included
US4.4	As a **Product Manager**, I want to see user activity metrics (logins, registrations) so that I can track adoption	Should-Have	- Dashboard shows daily/weekly/monthly active users - Breakdown by user type (student, teacher, etc.) - Exportable as CSV

Functional Requirements

FR4.1: Prometheus Metrics

- Instrument all services with Prometheus client libraries
- Expose metrics endpoint: `GET /metrics` (port 9090)
- Standard metrics:
  - **HTTP**: `http_requests_total`, `http_request_duration_seconds`, `http_request_size_bytes`
  - **System**: `process_cpu_seconds_total`, `process_resident_memory_bytes`, `nodejs_heap_size_bytes`
  - **Business**: `auth_login_total`, `auth_registration_total`, `bulk_import_jobs_total`, `kafka_messages_produced_total`
- Custom metrics:
  - `active_sessions_total` (gauge)
  - `password_reset_requests_total` (counter)
  - `bulk_import_rows_processed_total` (counter)
- Labels: `service`, `method`, `route`, `status_code`, `user_type`

FR4.2: Grafana Dashboards

- Pre-built dashboards:
  1. **System Overview**: CPU, memory, disk, network for all services
  2. **API Performance**: Request rate, latency (p50, p95, p99), error rate

3. **Authentication**: Login attempts, success rate, lockouts, active sessions
4. **Bulk Imports**: Jobs in progress, throughput, error rate
5. **Kafka**: Message rate, consumer lag, partition distribution
6. **Database**: Connection pool usage, query duration, slow queries

- Dashboard features:
  - Time range selector (last 1h, 6h, 24h, 7d)
  - Service filter dropdown
  - Auto-refresh (15s, 30s, 1m)
  - Annotations for deployments

FR4.3: Structured Logging (ELK Stack)

- Log format: JSON with standard fields:

```
{
  "timestamp": "2025-12-15T20:50:00Z",
  "level": "error",
  "service": "auth-service",
  "traceId": "abc-123",
  "spanId": "def-456",
  "message": "Login failed",
  "userId": "...",
  "userType": "student",
  "error": {
    "name": "InvalidCredentialsError",
    "message": "...",
    "stack": "..."
  },
  "context": {
    "ip": "192.168.1.1",
    "userAgent": "..."
  }
}
```

- Log levels: `error`, `warn`, `info`, `debug`
- Log shipping: Filebeat → Elasticsearch

- Kibana index patterns: `lms-logs-*`
- Retention: 30 days (configurable)

FR4.4: Distributed Tracing (OpenTelemetry)

- Instrument services with OpenTelemetry SDK
- Trace propagation: W3C Trace Context headers
- Spans:
  - HTTP requests (auto-instrumented)
  - Database queries (Prisma instrumentation)
  - Kafka produce/consume
  - Redis operations
- Export to Jaeger (or Tempo)
- Trace sampling: 10% in production, 100% in dev

FR4.5: Alerting

- Alert rules (Prometheus Alertmanager):
  1. **High Error Rate**: Error rate >5% for 5 minutes
  2. **High Latency**: p95 latency >1s for 5 minutes
  3. **Service Down**: Service unreachable for 1 minute
  4. **High Memory**: Memory usage >90% for 5 minutes
  5. **Kafka Consumer Lag**: Lag >1000 messages for 10 minutes
  6. **Database Connections**: Connection pool >80% for 5 minutes
- Alert channels:
  - Slack: `#lms-alerts` channel
  - Email: `devops@example.com`
  - PagerDuty: For critical alerts (P0/P1)
- Alert format:

```
[CRITICAL] auth-service: High Error Rate
Error rate: 8.5% (threshold: 5%)
Time: 2025-12-15 20:50:00
Runbook: https://wiki.example.com/runbooks/high-error-rate
Dashboard: https://grafana.example.com/d/auth-service
```

FR4.6: Health Checks

- Implement health check endpoints:
  - `GET /health`: Liveness probe (returns 200 if service running)
  - `GET /health/ready`: Readiness probe (checks DB, Redis, Kafka connectivity)
- Response format:

```
{
  "status": "healthy",
  "timestamp": "...",
  "checks": {
    "database": { "status": "up", "latency": "5ms" },
    "redis": { "status": "up", "latency": "2ms" },
    "kafka": { "status": "up", "latency": "10ms" }
  }
}
```

Non-Functional Requirements

NFR4.1: Performance

- Metrics collection overhead: <1% CPU
- Log shipping latency: <10 seconds
- Trace sampling overhead: <2% latency increase

NFR4.2: Reliability

- Monitoring stack uptime: >99.9%
- No data loss for logs/metrics (buffering on network issues)

NFR4.3: Scalability

- Support 100,000 metrics/second
- Elasticsearch handles 10 GB logs/day

Technical Architecture

New Services:

```
monitoring/
├── prometheus/
│   ├── prometheus.yml
│   ├── alerts.yml
│   └── Dockerfile
├── grafana/
│   ├── dashboards/
│   │   ├── system-overview.json
│   │   ├── api-performance.json
│   │   ├── authentication.json
│   │   └── bulk-imports.json
│   ├── provisioning/
│   │   ├── datasources/
│   │   │   └── prometheus.yml
│   │   └── dashboards/
│   │       └── default.yml
│   └── Dockerfile
├── elasticsearch/
│   ├── elasticsearch.yml
│   └── Dockerfile
├── kibana/
│   ├── kibana.yml
│   └── Dockerfile
├── filebeat/
│   ├── filebeat.yml
│   └── Dockerfile
├── jaeger/
│   └── docker-compose.jaeger.yml
└── alertmanager/
```

```
    ├── alertmanager.yml
    └── Dockerfile
```

Service Updates:

```
services/auth-service/
├── src/
│   ├── config/
│   │   ├── metrics.ts          [NEW] Prometheus client setup
│   │   ├── logger.ts           [NEW] Winston/Pino structured logging
│   │   └── tracing.ts          [NEW] OpenTelemetry setup
│   ├── middleware/
│   │   ├── metricsMiddleware.ts [NEW] HTTP metrics collection
│   │   └── tracingMiddleware.ts [NEW] Trace context propagation
│   └── routes/
│       └── health.route.ts      [NEW] Health check endpoints
```

Docker Compose Update:

```yaml
# docker-compose.yml (additions)
services:
  prometheus:
    image: prom/prometheus:latest
    volumes:
      - ./monitoring/prometheus:/etc/prometheus
      - prometheus-data:/prometheus
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana:latest
    volumes:
      - ./monitoring/grafana:/etc/grafana/provisioning
      - grafana-data:/var/lib/grafana
```

```yaml
    ports:
      - "3001:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.0
    environment:
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ports:
      - "9200:9200"

  kibana:
    image: docker.elastic.co/kibana/kibana:8.11.0
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch

  filebeat:
    image: docker.elastic.co/beats/filebeat:8.11.0
    volumes:
      - ./monitoring/filebeat/filebeat.yml:/usr/share/filebeat/filebeat.yml
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
    depends_on:
      - elasticsearch

  jaeger:
    image: jaegertracing/all-in-one:latest
    ports:
      - "16686:16686"  # UI
      - "4318:4318"    # OTLP HTTP
```

Dependencies & Tech Stack Updates

New Dependencies (per service):

- `prom-client`: Prometheus metrics
- `winston` or `pino`: Structured logging
- `@opentelemetry/sdk-node`: OpenTelemetry
- `@opentelemetry/auto-instrumentations-node`: Auto-instrumentation
- `@opentelemetry/exporter-jaeger`: Jaeger exporter

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V4.1	Prometheus metrics collection	Browser	- Access http://localhost:9090 - Query `http_requests_total` - See metrics from all services
V4.2	Grafana dashboards	Browser	- Access http://localhost:3001 - View “API Performance” dashboard - See real-time request rate graph
V4.3	Elasticsearch log ingestion	Kibana	- Access http://localhost:5601 - Search for recent logs - Filter by service, level, traceId
V4.4	Distributed tracing	Jaeger UI	- Access http://localhost:16686 - Search for traces - See end-to-end request flow (Kong → Auth → DB)
V4.5	Alert firing	Manual	- Trigger high error rate (return 500 from endpoint) - Alert sent to Slack within 1 minute - Alert contains correct details
V4.6	Health checks	Postman	- GET /health returns 200 - GET /health/ready returns 200 when all

ID	Test Case	Tool	Acceptance Criteria
			deps up - Returns 503 when DB down
V4.7	Log correlation	Kibana	- Make API request with trace ID - Search logs by trace ID - See all related logs (auth, kafka, notification)
V4.8	Dashboard accuracy	Manual	- Generate 100 login requests - Check Grafana “Authentication” dashboard - Verify count matches (100 logins)

Load Testing:

- Artillery: 1000 RPS for 5 minutes
- Expected: Metrics/logs/traces captured without data loss, monitoring stack stable

---

Phase 5: API Gateway Enhancement & Documentation

Phase Goals

1. Enhance Kong Gateway with advanced plugins (rate limiting, request transformation, caching)
2. Implement API versioning strategy
3. Complete OpenAPI/Swagger documentation for all endpoints
4. Add API key authentication for third-party integrations
5. Implement request/response logging at gateway level

User Stories

ID	User Story	Priority	Acceptance Criteria
US5.1	As a **Third-Party Developer**, I want to access LMS APIs using an API key so that I can build integrations without user credentials	Must-Have	- API key generation endpoint available - API key validated by Kong - Rate limits enforced per API key - Usage analytics available
US5.2	As a **Frontend Developer**, I want comprehensive API documentation so that I can integrate with backend services efficiently	Must-Have	- Swagger UI accessible at /api-docs - All endpoints documented with examples - Request/ response schemas defined - Authentication flow explained
US5.3	As a **Product Manager**, I want to release new API versions without breaking existing clients so that we can iterate quickly	Must-Have	- API versioning via URL path (/api/ v1, /api/v2) - Old versions supported for 6 months - Deprecation warnings in response headers

ID	User Story	Priority	Acceptance Criteria
US5.4	As a **DevOps Engineer**, I want Kong to cache frequently accessed endpoints so that we reduce database load	Should-Have	- Caching enabled for GET endpoints - Cache TTL configurable per route - Cache hit rate >70% for static data

Functional Requirements

FR5.1: API Key Management

- New endpoints in auth-service:
  - `POST /api/v1/auth/api-keys`: Generate API key (requires ADMIN or COLLEGE_ADMIN role)
  - `GET /api/v1/auth/api-keys`: List API keys
  - `DELETE /api/v1/auth/api-keys/{keyId}`: Revoke API key
- API key format: `lms_live_<random_32_chars>` (e.g., `lms_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6`)
- Store in database: `ApiKey` table with fields: `id, key (hashed), name, createdBy, scopes, rateLimit, expiresAt`
- Kong plugin: `key-auth` plugin configured
- Scopes: `read:courses`, `write:students`, `read:analytics`, etc.

FR5.2: Kong Plugin Configuration

- **Rate Limiting** (`rate-limiting` plugin):
  - Authenticated users: 1000 requests/hour
  - API keys: Configurable per key (default 100/hour)
  - Unauthenticated: 10 requests/hour
- **Request Transformer** (`request-transformer` plugin):
  - Add `X-Forwarded-For` header
  - Add `X-Request-ID` (UUID) for tracing

- **Response Transformer** (`response-transformer` plugin):
  - Add `X-RateLimit-Remaining` header
  - Add `X-API-Version` header
  - Add `Deprecation` header for deprecated endpoints
- **Proxy Caching** (`proxy-cache` plugin):
  - Cache GET requests for static data (courses, colleges, branches)
  - TTL: 5 minutes (configurable per route)
  - Cache key: `{method}:{path}:{query_params}`
  - Bypass cache with `Cache-Control: no-cache` header
- **CORS** (`cors` plugin):
  - Allow origins: `https://lms.example.com`, `https://admin.lms.example.com`
  - Allow methods: GET, POST, PUT, DELETE, PATCH
  - Allow headers: Authorization, Content-Type, X-Request-ID
  - Max age: 86400 seconds

FR5.3: API Versioning

- URL-based versioning: `/api/v1/...`, `/api/v2/...`
- Kong routes configured per version:

```
routes:
  - name: auth-v1
    paths: ["/api/v1/auth"]
    service: auth-service-v1
  - name: auth-v2
    paths: ["/api/v2/auth"]
    service: auth-service-v2  # Can point to same service with version
            header
```

- Version deprecation policy:
  - New version announced 3 months before release
  - Old version supported for 6 months after new release
  - Deprecation warning in response: `Deprecation: version="v1", sunset="2026-06-15"`
- Version detection in service:
  - Read `X-API-Version` header (set by Kong)
  - Route to appropriate controller/logic

FR5.4: OpenAPI Documentation

- Complete OpenAPI 3.0 spec for all endpoints
- Swagger UI at `/api-docs` (served by auth-service or separate docs service)
- Documentation includes:
  - Endpoint descriptions
  - Request/response schemas (with examples)
  - Authentication requirements
  - Error responses
  - Rate limits
- Auto-generate from code annotations (using `tsoa` or `swagger-jsdoc`)
- Versioned docs: `/api-docs/v1`, `/api-docs/v2`

FR5.5: Request/Response Logging

- Kong plugin: `file-log` or `http-log` plugin
- Log format:

```
{
  "requestId": "...",
  "method": "POST",
  "path": "/api/v1/auth/student/login",
  "statusCode": 200,
  "latency": 150,
  "requestSize": 256,
  "responseSize": 512,
  "clientIp": "192.168.1.1",
  "userAgent": "...",
  "timestamp": "..."
}
```

- Send to Kafka topic: `gateway.access.logs`
- Consumed by logging service → Elasticsearch

FR5.6: API Analytics

- Track API usage metrics:
  - Requests per endpoint

    - Requests per API key
    - Error rate per endpoint
    - Latency percentiles
- Store in database or time-series DB (InfluxDB)
- Dashboard in Grafana: “API Usage Analytics”

Non-Functional Requirements

NFR5.1: Performance

- Kong latency overhead: <10ms
- Cache hit rate: >70% for cacheable endpoints
- API documentation page load: <2 seconds

NFR5.2: Security

- API keys hashed in database (SHA-256)
- HTTPS enforced (HTTP redirects to HTTPS)
- CORS properly configured (no wildcard origins in production)

NFR5.3: Availability

- Kong uptime: >99.9%
- Documentation always accessible (even if backend down)

Technical Architecture

Kong Configuration Updates:

```
gateway/kong/
├── kong.yaml                    [UPDATED] Add plugins, versioned routes
├── plugins/
│   ├── paseto-vault-auth-lua/   [EXISTING]
│   └── api-key-validator-lua/   [NEW] Custom API key validation
└── declarative/
    ├── v1-routes.yaml           [NEW] Version 1 routes
    └── v2-routes.yaml           [NEW] Version 2 routes (future)
```

Kong Configuration Example:

```yaml
# kong.yaml (excerpt)
services:
  - name: auth-service-v1
    url: http://auth-service:3000
    routes:
      - name: auth-v1
        paths: ["/api/v1/auth"]
        strip_path: false
        plugins:
          - name: rate-limiting
            config:
              minute: 100
              hour: 1000
              policy: redis
              redis_host: redis
          - name: request-transformer
            config:
              add:
                headers:
                  - "X-API-Version: v1"
                  - "X-Request-ID: $(uuid)"
          - name: response-transformer
            config:
              add:
                headers:
                  - "X-RateLimit-Remaining: $(ratelimit.remaining)"
          - name: cors
            config:
              origins: ["https://lms.example.com"]
              methods: ["GET", "POST", "PUT", "DELETE"]
              headers: ["Authorization", "Content-Type"]
          - name: proxy-cache
            config:
```

```
        strategy: memory
        content_type: ["application/json"]
        cache_ttl: 300
        cache_control: true
```

Auth Service Updates:

```
services/auth-service/
├── src/
│   ├── controllers/
│   │   └── api-key.controller.ts    [NEW]
│   ├── services/
│   │   └── api-key.service.ts       [NEW]
│   ├── routes/
│   │   └── api-key.route.ts         [NEW]
│   └── middleware/
│       └── apiKeyValidator.ts       [NEW] Validate API key scopes
├── prisma/
│   └── schema.prisma                [UPDATED] Add ApiKey model
└── docs/
    └── swagger/
        ├── openapi.yaml             [UPDATED] Complete all endpoints
        └── v2/
            └── openapi.yaml         [NEW] Version 2 spec (future)
```

Data Model:

```
model ApiKey {
  id          String   @id @default(uuid())
  keyHash     String   @unique  // SHA-256 hash of API key
  name        String            // "Mobile App Integration"
  description String?
  createdBy   String
  createdByType String          // "college", "organization"
```

```
  scopes      String[]           // ["read:courses", "write:students"]
  rateLimit   Int      @default(100)  // Requests per hour
  isActive    Boolean  @default(true)
  expiresAt   DateTime?
  lastUsedAt  DateTime?
  createdAt   DateTime @default(now())

  @@index([keyHash])
  @@index([createdBy])
}

model ApiKeyUsage {
  id         String   @id @default(uuid())
  apiKeyId   String
  endpoint   String
  method     String
  statusCode Int
  latency    Int      // milliseconds
  timestamp  DateTime @default(now())

  @@index([apiKeyId, timestamp])
  @@index([endpoint])
}
```

Dependencies & Tech Stack Updates

New Dependencies:

- `tsoa`: TypeScript OpenAPI generator (or `swagger-jsdoc`)
- `swagger-ui-express`: Serve Swagger UI
- Kong plugins (built-in): `rate-limiting`, `cors`, `proxy-cache`, `key-auth`

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V5.1	API key generation	Postman	- POST /api-keys returns new key - Key format: `lms_live_<32_chars>` - Key stored hashed in DB
V5.2	API key authentication	Postman	- Request with valid API key succeeds - Request with invalid key returns 401 - Request without key returns 401
V5.3	Rate limiting	Artillery	- Send 101 requests in 1 minute - 101st request returns 429 - Retry-After header present
V5.4	API versioning	Postman	- /api/v1/auth/login works - /api/v2/auth/login returns 404 (not implemented yet) - Response includes X-API-Version header
V5.5	Swagger documentation	Browser	- Access /api-docs - See all endpoints listed - “Try it out” feature works - Schemas have examples
V5.6	Proxy caching	Postman	- GET /api/v1/courses (1st request) - Check X-Cache-Status header (MISS) - GET /api/v1/courses (2nd request) - Check X-Cache-Status header (HIT)
V5.7	CORS headers	Browser	- Fetch from https://lms.example.com - CORS headers present - Request succeeds
V5.8	Request logging	Kafka UI	- Make API request - Check `gateway.access.logs` topic - Log contains request details

Performance Testing:

- Artillery: 500 RPS for 5 minutes with caching enabled
- Expected: Cache hit rate >70%, p95 latency <200ms

End of Phases 1-5

Next Steps:

- Proceed to 02-PHASES-06-10.md for Core LMS Features (Course Management, Enrollment, CMS, Attendance, Assignments)

LMS Platform PRD - Phases 6-9

Core LMS Features - Part 1

Phase 6: Course Management Service

Phase Goals

1. Create new microservice for course lifecycle management
2. Implement course CRUD operations with curriculum versioning
3. Establish course-subject-teacher relationships
4. Add course prerequisites and dependency management
5. Integrate with existing auth service for role-based access

User Stories

ID	User Story	Priority	Acceptance Criteria
US6.1	As an **HOD**, I want to create a new course with subjects so that I can define the curriculum for my department	Must-Have	- Course created with name, code, credits, description - Multiple subjects added to course - Subjects have syllabus, credits, type (theory/lab) - Course published to catalog
		Must-Have	- Teacher assigned to subject-section

ID	User Story	Priority	Acceptance Criteria
US6.2	As a **Teacher**, I want to be assigned to teach specific subjects so that students know who their instructor is		combination - One subject can have multiple teachers (different sections) - Assignment notification sent to teacher - Visible in course catalog
US6.3	As a **Dean**, I want to set course prerequisites so that students take courses in proper sequence	Should-Have	- Prerequisites defined as course IDs - Enrollment validation checks prerequisites - Clear error message if prerequisites not met - Override option for special cases
US6.4	As a **College Admin**, I want to version course curricula so that I can track changes over academic years	Should-Have	- Course versions linked to academic year - Students enrolled in specific version - Version comparison view - Cannot delete version with active enrollments

Functional Requirements

FR6.1: Course CRUD Operations

- Endpoints:
  - `POST /api/v1/courses`: Create course (HOD/Dean only)
  - `GET /api/v1/courses`: List courses (with filters: branch, year, active)
  - `GET /api/v1/courses/{id}`: Get course details

    - PUT /api/v1/courses/{id}: Update course
    - DELETE /api/v1/courses/{id}: Soft delete course
- Course fields: name, code, branchId, credits, description, type (core/elective), isActive, academicYear
- Validation: Unique course code per branch, credits > 0

FR6.2: Subject Management

- Endpoints:
    - POST /api/v1/courses/{id}/subjects: Add subject to course
    - GET /api/v1/courses/{id}/subjects: List subjects
    - PUT /api/v1/subjects/{id}: Update subject
    - DELETE /api/v1/subjects/{id}: Remove subject
- Subject fields: name, code, credits, type (theory/lab/project), syllabus, learningOutcomes

FR6.3: Teacher-Subject Assignment

- Endpoints:
    - POST /api/v1/subjects/{id}/assign-teacher: Assign teacher
    - GET /api/v1/teachers/{id}/subjects: Get teacher's subjects
    - DELETE /api/v1/subjects/{id}/teachers/{teacherId}: Unassign
- Assignment includes: teacherId, subjectId, sectionId, academicYear, semester
- Publish Kafka event: course.teacher.assigned

FR6.4: Prerequisites Management

- Endpoints:
    - POST /api/v1/courses/{id}/prerequisites: Add prerequisite
    - GET /api/v1/courses/{id}/prerequisites: List prerequisites
    - DELETE /api/v1/courses/{id}/prerequisites/{prereqId}: Remove
- Validation: No circular dependencies, prerequisite course exists

FR6.5: Course Catalog API

- GET /api/v1/catalog/courses: Public course catalog
- Filters: branch, year, type, credits, hasSeats
- Response includes: course details, subjects, assigned teachers, available seats
- Pagination: 20 courses per page

Non-Functional Requirements

NFR6.1: Performance

- Course list API: <200ms (p95)
- Course creation: <500ms
- Catalog search: <300ms with 1000+ courses

NFR6.2: Data Integrity

- Foreign key constraints (courseId, branchId, teacherId)
- Transaction support for multi-subject course creation
- Soft delete to preserve historical data

NFR6.3: Scalability

- Support 10,000+ courses
- 100+ concurrent course creations (bulk import)

Technical Architecture

New Service Structure:

```
services/course-service/
├── Dockerfile
├── package.json
├── tsconfig.json
├── prisma/
│   ├── schema.prisma
│   └── migrations/
├── src/
│   ├── index.ts
│   ├── app.ts
│   ├── config/
│   │   ├── env.ts
│   │   ├── database.ts
│   │   └── kafka.ts
```

```
|   ├── controllers/
|   |   ├── course.controller.ts
|   |   ├── subject.controller.ts
|   |   ├── assignment.controller.ts
|   |   └── catalog.controller.ts
|   ├── services/
|   |   ├── course.service.ts
|   |   ├── subject.service.ts
|   |   ├── prerequisite.service.ts
|   |   └── validation.service.ts
|   ├── routes/
|   |   ├── course.route.ts
|   |   ├── subject.route.ts
|   |   └── catalog.route.ts
|   ├── middleware/
|   |   ├── auth.middleware.ts
|   |   └── rbac.middleware.ts
|   ├── types/
|   |   ├── course.types.ts
|   |   └── subject.types.ts
|   ├── utils/
|   |   ├── prerequisite-validator.ts
|   |   └── circular-dependency-checker.ts
|   └── messaging/
|       └── producer.ts
└── tests/
    ├── unit/
    └── integration/
```

Data Models (Prisma):

```
model Course {
  id          String   @id @default(uuid())
  code        String   @unique
```

```
  name          String
  branchId      String
  credits       Int
  description   String?
  type          String   // "core", "elective"
  academicYear  String
  semester      Int?
  isActive      Boolean  @default(true)
  createdBy     String
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
  deletedAt     DateTime?

  subjects      Subject[]
  prerequisites CoursePrerequisite[] @relation("CoursePrerequisites")
  prerequisiteFor CoursePrerequisite[] @relation("PrerequisiteFor")

  @@index([branchId, academicYear])
  @@index([isActive])
}

model Subject {
  id              String   @id @default(uuid())
  courseId        String
  code            String
  name            String
  credits         Int
  type            String   // "theory", "lab", "project"
  syllabus        String?  @db.Text
  learningOutcomes String? @db.Text
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  course          Course   @relation(fields: [courseId], references: [id])
```

```
  teacherAssignments TeacherSubjectAssignment[]

  @@unique([courseId, code])
  @@index([courseId])
}

model TeacherSubjectAssignment {
  id           String   @id @default(uuid())
  teacherId    String
  subjectId    String
  sectionId    String
  academicYear String
  semester     Int
  assignedAt   DateTime @default(now())
  assignedBy   String

  subject      Subject  @relation(fields: [subjectId], references: [id])

  @@unique([teacherId, subjectId, sectionId, academicYear, semester])
  @@index([teacherId])
  @@index([subjectId])
}

model CoursePrerequisite {
  id             String   @id @default(uuid())
  courseId       String
  prerequisiteId String
  isStrict       Boolean  @default(true)
  createdAt      DateTime @default(now())

  course         Course   @relation("CoursePrerequisites", fields: [courseId],
references: [id])
  prerequisite   Course   @relation("PrerequisiteFor", fields: [prerequisiteId],
references: [id])
```

```
  @@unique([courseId, prerequisiteId])
}
```

Kong Gateway Update:

```
# gateway/kong/kong.yaml
services:
  - name: course-service
    url: http://course-service:3001
    routes:
      - name: courses-api
        paths: ["/api/v1/courses", "/api/v1/subjects", "/api/v1/catalog"]
        plugins:
          - name: paseto-vault-auth
          - name: rate-limiting
            config:
              minute: 100
```

Kafka Topics:

- `course.created`
- `course.updated`
- `course.teacher.assigned`
- `subject.created`

Dependencies & Tech Stack

- Express.js, TypeScript, Prisma
- PostgreSQL (shared with auth-service or separate DB)
- Kafka (existing infrastructure)
- Redis (caching course catalog)

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V6.1	Create course with subjects	Postman	- Course created successfully - 3 subjects added - Returns 201 with course ID
V6.2	Assign teacher to subject	Postman	- Teacher assigned to subject-section - Kafka event published - Visible in GET /teachers/{id}/subjects
V6.3	Prerequisite validation	Postman	- Add prerequisite successfully - Circular dependency rejected - Returns 400 with clear error
V6.4	Course catalog API	Postman	- Returns paginated courses - Filters work (branch, type) - Response time <300ms
V6.5	Bulk course creation	Jest	- Import 100 courses via script - All created successfully - No duplicate codes
V6.6	Soft delete course	Postman	- DELETE sets deletedAt - Course not in active list - Historical data preserved

---

Phase 7: Student Enrollment & Allocation

Phase Goals

1. Implement student enrollment workflow
2. Auto-allocate students to sections based on capacity
3. Manage waitlists for full courses
4. Handle enrollment conflicts (timetable clashes)

5. Send enrollment confirmations

User Stories

ID	User Story	Priority	Acceptance Criteria
US7.1	As a **Student**, I want to enroll in courses for the semester so that I can attend classes	Must-Have	- Select courses from catalog - See available seats - Enroll in multiple courses - Receive confirmation email
US7.2	As a **College Admin**, I want students auto-allocated to sections so that sections are balanced	Must-Have	- Auto-allocation runs after enrollment deadline - Sections filled evenly - Students notified of section assignment - Manual override available
US7.3	As a **Student**, I want to join a waitlist if a course is full so that I can enroll if seats open up	Should-Have	- Waitlist position shown - Auto-enrolled when seat available - Notification sent - Can leave waitlist
US7.4	As an **HOD**, I want to see enrollment statistics so that I can plan resources	Should-Have	- Dashboard shows enrollment count per course - Section-wise breakdown - Waitlist counts - Export to CSV

Functional Requirements

FR7.1: Enrollment API

- Endpoints:
  - `POST /api/v1/enrollments`: Enroll in course
  - `GET /api/v1/students/{id}/enrollments`: Get student enrollments
  - `DELETE /api/v1/enrollments/{id}`: Drop course (before deadline)
  - `GET /api/v1/courses/{id}/enrollments`: Get course enrollments (admin)
- Enrollment validation:
  - Prerequisites met
  - No timetable conflicts
  - Seats available or waitlist
  - Enrollment window open
  - Credit limit not exceeded (max 24 credits/semester)

FR7.2: Section Auto-Allocation

- Algorithm:
  1. Group students by course
  2. Sort sections by capacity
  3. Distribute students evenly (round-robin)
  4. Consider preferences (if provided)
  5. Handle constraints (lab groups, reserved seats)
- Endpoint: `POST /api/v1/admin/allocate-sections` (manual trigger)
- Scheduled job: Runs automatically after enrollment deadline
- Publish events: `enrollment.section.allocated`

FR7.3: Waitlist Management

- Endpoints:
  - `POST /api/v1/courses/{id}/waitlist`: Join waitlist
  - `GET /api/v1/students/{id}/waitlist`: Get waitlist entries
  - `DELETE /api/v1/waitlist/{id}`: Leave waitlist
- Waitlist processing:
  - When seat available, auto-enroll first in waitlist
  - Send notification to student (24-hour acceptance window)
  - If not accepted, offer to next in waitlist

FR7.4: Enrollment Statistics

- Endpoints:
  - `GET /api/v1/analytics/enrollment-stats`: Overall stats
  - `GET /api/v1/courses/{id}/stats`: Course-specific stats
- Metrics: total enrolled, section-wise count, waitlist count, drop rate
- Export: CSV download

FR7.5: Enrollment Periods

- Endpoints:
  - `POST /api/v1/admin/enrollment-periods`: Create period
  - `GET /api/v1/enrollment-periods/current`: Get active period
- Period fields: `startDate, endDate, academicYear, semester, type (regular/late)`
- Validation: Enrollment only during active period

Non-Functional Requirements

NFR7.1: Performance

- Enrollment API: <500ms
- Auto-allocation: Process 5000 students in <5 minutes
- Waitlist processing: <10 seconds per student

NFR7.2: Concurrency

- Handle 500 concurrent enrollments
- Optimistic locking for seat availability
- Race condition handling (last seat scenario)

NFR7.3: Data Integrity

- Transactional enrollment (all-or-nothing for multiple courses)
- Seat count accuracy (no overbooking)

Technical Architecture

Course Service Updates:

```
services/course-service/
├── src/
│   ├── controllers/
│   │   ├── enrollment.controller.ts      [NEW]
│   │   └── waitlist.controller.ts        [NEW]
│   ├── services/
│   │   ├── enrollment.service.ts         [NEW]
│   │   ├── allocation.service.ts         [NEW]
│   │   ├── waitlist.service.ts           [NEW]
│   │   └── conflict-detector.service.ts  [NEW]
│   ├── utils/
│   │   ├── allocation-algorithm.ts       [NEW]
│   │   └── credit-calculator.ts          [NEW]
│   ├── jobs/
│   │   ├── auto-allocate.job.ts          [NEW]
│   │   └── waitlist-processor.job.ts     [NEW]
│   └── routes/
│       └── enrollment.route.ts           [NEW]
```

Data Models:

```
model Enrollment {
  id            String   @id @default(uuid())
  studentId     String
  courseId      String
  sectionId     String?  // Null until allocated
  academicYear  String
  semester      Int
  status        String   // "enrolled", "waitlisted", "dropped"
  enrolledAt    DateTime @default(now())
  allocatedAt   DateTime?
  droppedAt     DateTime?
  grade         String?
```

```
  @@unique([studentId, courseId, academicYear, semester])
  @@index([studentId])
  @@index([courseId])
  @@index([status])
}

model Waitlist {
  id          String   @id @default(uuid())
  studentId   String
  courseId    String
  position    Int
  joinedAt    DateTime @default(now())
  notifiedAt  DateTime?
  expiresAt   DateTime?

  @@unique([studentId, courseId])
  @@index([courseId, position])
}

model EnrollmentPeriod {
  id          String   @id @default(uuid())
  academicYear String
  semester    Int
  type        String   // "regular", "late", "add_drop"
  startDate   DateTime
  endDate     DateTime
  isActive    Boolean  @default(true)

  @@index([isActive, startDate, endDate])
}
```

Kafka Topics:

- `enrollment.created`
- `enrollment.section.allocated`

- `enrollment.dropped`
- `waitlist.joined`
- `waitlist.promoted`

Dependencies & Tech Stack

- Bull or Agenda (job scheduling for auto-allocation)
- Redis (distributed locking for seat allocation)

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V7.1	Student enrollment	Postman	- Enroll in 5 courses - Prerequisites validated - Credit limit enforced - Confirmation email sent
V7.2	Section auto-allocation	Script	- 1000 students enrolled - Run allocation - All students assigned sections - Sections balanced (±10%)
V7.3	Waitlist functionality	Postman	- Enroll in full course - Added to waitlist - Position shown - Promoted when seat opens
V7.4	Concurrent enrollment	Artillery	- 100 students enroll in same course simultaneously - Seat count accurate - No overbooking
V7.5	Enrollment statistics	Postman	- GET stats returns correct counts - Matches database query - CSV export works

---

Phase 8: Content Management System (CMS)

Phase Goals

1. Build content upload and storage system using S3
2. Implement presigned URL generation for secure access
3. Add content versioning and metadata management
4. Implement role-based access control for content
5. Support multiple file types (PDF, video, images, documents)

User Stories

ID	User Story	Priority	Acceptance Criteria
US8.1	As a **Teacher**, I want to upload lecture notes (PDF) so that students can access them	Must-Have	- Upload PDF up to 50MB - File stored in S3 - Students can download - Upload progress shown
US8.2	As a **Teacher**, I want to upload video lectures so that students can watch them online	Must-Have	- Upload video up to 500MB - Presigned URL for streaming - Video metadata stored (duration, size) - Thumbnail generated
US8.3	As a **Student**, I want to access course content only if I’m enrolled so that content is protected	Must-Have	- Access denied if not enrolled - Presigned URL expires in 1 hour

ID	User Story	Priority	Acceptance Criteria
			- Download tracked (analytics)
US8.4	As a **Teacher**, I want to organize content into folders (modules/weeks) so that it's easy to navigate	Should-Have	- Create folders - Nested folder support - Drag-and-drop reorder - Folder permissions

Functional Requirements

FR8.1: File Upload

- Endpoints:
  - `POST /api/v1/content/upload`: Initiate upload (returns presigned POST URL)
  - `POST /api/v1/content/confirm`: Confirm upload completion
  - `PUT /api/v1/content/{id}`: Update metadata
  - `DELETE /api/v1/content/{id}`: Delete content (soft delete)
- Upload flow:
  1. Client requests upload URL
  2. Server generates S3 presigned POST URL
  3. Client uploads directly to S3
  4. Client confirms upload
  5. Server saves metadata to database
- Supported types: PDF, DOCX, PPTX, MP4, MP3, JPG, PNG (configurable)
- Size limits: Documents 50MB, Videos 500MB, Images 10MB

FR8.2: Content Access

- Endpoints:
  - `GET /api/v1/content/{id}/download`: Get presigned download URL
  - `GET /api/v1/courses/{id}/content`: List course content
  - `GET /api/v1/content/{id}/stream`: Get streaming URL (videos)

- Access control:
  - Teachers: Full access to their course content
  - Students: Access only if enrolled
  - Admins: Full access
- Presigned URL expiry: 1 hour (configurable)
- Track downloads: Log to Kafka for analytics

FR8.3: Content Metadata

- Metadata fields:
  - `title, description, type, size, mimeType, duration (videos), uploadedBy, courseId, subjectId, folderId`
  - `isPublished, publishedAt, version, tags`
- Search/filter: By course, subject, type, tags, date range

FR8.4: Folder/Module Organization

- Endpoints:
  - `POST /api/v1/courses/{id}/folders`: Create folder
  - `GET /api/v1/courses/{id}/folders`: List folders
  - `PUT /api/v1/folders/{id}`: Update folder
  - `POST /api/v1/content/{id}/move`: Move content to folder
- Folder structure: Course → Modules → Weeks → Content
- Order: Manual ordering (position field)

FR8.5: Content Versioning

- Upload new version of existing content
- Keep previous versions (S3 versioning)
- Endpoint: `POST /api/v1/content/{id}/versions`
- Students always see latest published version
- Teachers can access all versions

Non-Functional Requirements

NFR8.1: Performance

- Presigned URL generation: <100ms
- Content list API: <300ms
- Direct S3 upload (no server bottleneck)

NFR8.2: Storage

- S3 bucket: `lms-content-{env}`
- Folder structure: `{collegeId}/{courseId}/{contentId}/{filename}`
- Lifecycle policy: Archive to Glacier after 2 years

NFR8.3: Security

- Presigned URLs with expiry
- S3 bucket not publicly accessible
- Virus scanning for uploads (optional, ClamAV)

NFR8.4: Scalability

- Support 100,000+ files
- 1000 concurrent downloads

Technical Architecture

New Service:

```
services/content-service/
├── Dockerfile
├── package.json
├── src/
│   ├── controllers/
│   │   ├── upload.controller.ts
│   │   ├── content.controller.ts
│   │   └── folder.controller.ts
│   ├── services/
│   │   ├── s3.service.ts
│   │   ├── content.service.ts
│   │   ├── access-control.service.ts
│   │   └── thumbnail.service.ts (optional)
│   ├── middleware/
│   │   ├── file-validation.middleware.ts
│   │   └── enrollment-check.middleware.ts
```

```
|   ├── utils/
|   |   ├── presigned-url.ts
|   |   └── mime-type-validator.ts
|   └── config/
|       └── s3.config.ts
├── prisma/
|   └── schema.prisma
└── tests/
```

Data Models:

```
model Content {
  id          String   @id @default(uuid())
  title       String
  description String?
  type        String   // "document", "video", "image", "audio"
  mimeType    String
  size        Int      // bytes
  duration    Int?     // seconds (for videos/audio)
  s3Key       String   @unique
  s3Bucket    String
  courseId    String
  subjectId   String?
  folderId    String?
  uploadedBy  String
  uploadedByType String
  isPublished Boolean  @default(false)
  publishedAt DateTime?
  version     Int      @default(1)
  tags        String[]
  position    Int?     // For ordering
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  deletedAt   DateTime?
```

```
  @@index([courseId])
  @@index([folderId])
  @@index([isPublished])
}

model Folder {
  id          String   @id @default(uuid())
  name        String
  courseId    String
  parentId    String?  // For nested folders
  position    Int
  createdBy   String
  createdAt   DateTime @default(now())

  @@index([courseId])
  @@index([parentId])
}

model ContentAccess {
  id          String   @id @default(uuid())
  contentId   String
  userId      String
  userType    String
  action      String   // "view", "download"
  accessedAt  DateTime @default(now())
  ipAddress   String?

  @@index([contentId])
  @@index([userId])
}
```

S3 Configuration:

```
// src/config/s3.config.ts
export const s3Config = {
  bucket: process.env.S3_BUCKET_NAME,
  region: process.env.AWS_REGION,
  presignedUrlExpiry: 3600, // 1 hour
  maxFileSize: {
    document: 50 * 1024 * 1024, // 50MB
    video: 500 * 1024 * 1024,   // 500MB
    image: 10 * 1024 * 1024     // 10MB
  }
};
```

Kong Gateway Update:

```
services:
  - name: content-service
    url: http://content-service:3002
    routes:
      - name: content-api
        paths: ["/api/v1/content", "/api/v1/folders"]
```

Kafka Topics:

- `content.uploaded`
- `content.accessed`
- `content.deleted`

Dependencies & Tech Stack

- `@aws-sdk/client-s3`: S3 operations
- `@aws-sdk/s3-request-presigner`: Presigned URLs
- `multer`: File upload handling (for metadata)
- `sharp`: Image thumbnail generation (optional)
- `fluent-ffmpeg`: Video thumbnail generation (optional)

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V8.1	Upload PDF document	Postman	- Get presigned URL - Upload to S3 - Confirm upload - File accessible via download URL
V8.2	Upload video lecture	Postman	- Upload 100MB video - Metadata saved (duration, size) - Streaming URL works
V8.3	Access control	Postman	- Enrolled student can access - Non-enrolled student gets 403 - Presigned URL expires after 1 hour
V8.4	Folder organization	Postman	- Create folder structure - Move content to folders - List content by folder
V8.5	Content versioning	Postman	- Upload new version - Previous version accessible - Students see latest version
V8.6	Large file upload	Manual	- Upload 500MB video - Upload completes successfully - No server memory issues

---

Phase 9: Attendance Service

Phase Goals

1. Create attendance marking system with multiple methods (manual, QR code, geofencing)
2. Implement attendance analytics and reports
3. Add automated attendance notifications

4. Support makeup classes and attendance regularization
5. Integrate with timetable for automatic session creation

User Stories

ID	User Story	Priority	Acceptance Criteria
US9.1	As a **Teacher**, I want to mark attendance for my class so that student presence is recorded	Must-Have	- Select class session - Mark students present/absent - Save attendance - Students notified
US9.2	As a **Student**, I want to mark my attendance via QR code so that it’s quick and contactless	Should-Have	- Teacher generates QR code - Student scans QR - Attendance marked automatically - QR expires after class time
US9.3	As a **Student**, I want to see my attendance percentage so that I know if I meet minimum requirements	Must-Have	- View overall attendance % - Subject-wise breakdown - Warning if below 75% - Projected attendance
US9.4	As an **HOD**, I want attendance reports for my department so that I can track student engagement	Must-Have	- Generate reports (date range, subject, section) - Export to CSV/PDF - Defaulter list (below threshold)

ID	User Story	Priority	Acceptance Criteria
			- Email reports to stakeholders

User Stories (continued)

ID	User Story	Priority	Acceptance Criteria
US9.5	As a **Student**, I want to apply for attendance regularization if I was marked absent incorrectly	Should-Have	- Submit regularization request with reason - Teacher approves/ rejects - Attendance updated if approved - Notification sent

Functional Requirements

FR9.1: Attendance Marking

- Endpoints:
  - `POST /api/v1/attendance/sessions`: Create attendance session
  - `POST /api/v1/attendance/sessions/{id}/mark`: Mark attendance (bulk)
  - `PUT /api/v1/attendance/{id}`: Update single attendance record
  - `GET /api/v1/attendance/sessions/{id}`: Get session attendance
- Attendance status: `present, absent, late, excused`
- Bulk marking: Array of `{ studentId, status }`
- Validation: Session exists, teacher authorized, within time window

FR9.2: QR Code Attendance

- Endpoints:
  - `POST /api/v1/attendance/sessions/{id}/qr`: Generate QR code
  - `POST /api/v1/attendance/scan`: Student scans QR

- QR code contains: `sessionId, timestamp, signature (HMAC)`
- Expiry: QR valid for class duration + 15 minutes
- Validation: QR not expired, student enrolled, not already marked
- Geofencing (optional): Check student location within campus radius

FR9.3: Attendance Analytics

- Endpoints:
  - `GET /api/v1/students/{id}/attendance`: Student attendance summary
  - `GET /api/v1/courses/{id}/attendance`: Course attendance summary
  - `GET /api/v1/attendance/defaulters`: List students below threshold
- Metrics: total classes, attended, percentage, trend
- Threshold: Configurable (default 75%)
- Projected attendance: Predict final % based on remaining classes

FR9.4: Attendance Reports

- Endpoints:
  - `GET /api/v1/reports/attendance`: Generate report
  - `POST /api/v1/reports/attendance/export`: Export to CSV/PDF
- Filters: date range, course, section, subject, teacher
- Report types: summary, detailed, defaulter list
- Scheduled reports: Weekly email to HOD/Dean

FR9.5: Regularization Workflow

- Endpoints:
  - `POST /api/v1/attendance/regularization`: Submit request
  - `GET /api/v1/attendance/regularization`: List requests (teacher/student)
  - `PUT /api/v1/attendance/regularization/{id}`: Approve/reject
- Request fields: `attendanceId, reason, supportingDocuments (S3 URLs)`
- Approval workflow: Student → Teacher → HOD (if needed)
- Notification on approval/rejection

FR9.6: Timetable Integration

- Auto-create attendance sessions from timetable
- Scheduled job: Create sessions for next day
- Session fields: `courseId, subjectId, sectionId, teacherId, scheduledAt, duration`

Non-Functional Requirements

NFR9.1: Performance

- Mark attendance (50 students): <2 seconds
- QR scan response: <500ms
- Attendance report generation: <5 seconds for 1000 records

NFR9.2: Accuracy

- No duplicate attendance marks
- Timestamp accuracy (server time, not client)
- Geofencing accuracy: ±50 meters (if implemented)

NFR9.3: Scalability

- Support 10,000 attendance sessions/day
- 50,000 students marking attendance concurrently

NFR9.4: Availability

- Offline mode for teachers (sync later)
- QR code generation works even if DB slow

Technical Architecture

New Service:

```
services/attendance-service/
├── Dockerfile
├── package.json
├── src/
│   ├── controllers/
│   │   ├── attendance.controller.ts
│   │   ├── qr.controller.ts
│   │   ├── analytics.controller.ts
│   │   ├── report.controller.ts
│   │   └── regularization.controller.ts
│   ├── services/
```

```
│   │   ├── attendance.service.ts
│   │   ├── qr.service.ts
│   │   ├── geofencing.service.ts (optional)
│   │   ├── analytics.service.ts
│   │   ├── report.service.ts
│   │   └── regularization.service.ts
│   ├── utils/
│   │   ├── qr-generator.ts
│   │   ├── qr-validator.ts
│   │   ├── attendance-calculator.ts
│   │   └── pdf-generator.ts
│   ├── jobs/
│   │   ├── session-creator.job.ts
│   │   └── weekly-report.job.ts
│   └── routes/
├── prisma/
│   └── schema.prisma
└── tests/
```

Data Models:

```
model AttendanceSession {
  id          String   @id @default(uuid())
  courseId    String
  subjectId   String
  sectionId   String
  teacherId   String
  scheduledAt DateTime
  duration    Int      // minutes
  qrCode      String?  @unique
  qrExpiresAt DateTime?
  location    Json?    // { lat, lng, radius }
  status      String   // "scheduled", "ongoing", "completed"
  createdAt   DateTime @default(now())
```

```
  attendances Attendance[]

  @@index([courseId, scheduledAt])
  @@index([teacherId])
}

model Attendance {
  id          String   @id @default(uuid())
  sessionId   String
  studentId   String
  status      String   // "present", "absent", "late", "excused"
  markedAt    DateTime @default(now())
  markedBy    String?  // teacherId or "system" (for QR)
  method      String   // "manual", "qr", "geofence"
  location    Json?    // { lat, lng } if geofenced

  session     AttendanceSession @relation(fields: [sessionId], references: [id])
  regularization AttendanceRegularization?

  @@unique([sessionId, studentId])
  @@index([studentId])
  @@index([sessionId])
}

model AttendanceRegularization {
  id           String   @id @default(uuid())
  attendanceId String   @unique
  studentId    String
  reason       String   @db.Text
  documents    String[] // S3 URLs
  status       String   // "pending", "approved", "rejected"
  submittedAt  DateTime @default(now())
  reviewedBy   String?
  reviewedAt   DateTime?
```

```
  comments      String?

  attendance    Attendance @relation(fields: [attendanceId], references: [id])

  @@index([studentId, status])
}
```

Kafka Topics:

- `attendance.marked`
- `attendance.session.created`
- `attendance.regularization.submitted`
- `attendance.regularization.approved`
- `attendance.below.threshold` (trigger notification)

Dependencies & Tech Stack

- `qrcode`: QR code generation
- `jsqr`: QR code scanning (client-side)
- `pdfkit`: PDF report generation
- `geolib`: Geofencing calculations (optional)
- `bull`: Job scheduling for reports

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V9.1	Manual attendance marking	Postman	- Teacher marks 50 students - All saved correctly - Response time <2s
V9.2	QR code attendance	Mobile app/ Postman	- Generate QR code - Student scans - Attendance marked - QR expires after class
	Attendance analytics	Postman	

ID	Test Case	Tool	Acceptance Criteria
V9.3			- Student views attendance - Shows correct percentage - Warning if below 75%
V9.4	Attendance report	Postman	- Generate report for course - Export to CSV - Data matches database
V9.5	Regularization workflow	Postman	- Student submits request - Teacher approves - Attendance updated - Notifications sent
V9.6	Geofencing (if implemented)	Mobile app	- Student outside campus - QR scan rejected - Error message shown
V9.7	Concurrent QR scans	Artillery	- 100 students scan simultaneously - All marked correctly - No duplicates
V9.8	Defaulter list	Postman	- Generate defaulter list - Only students <75% shown - Sorted by percentage

---

End of Phases 6-9

Next Document: 03-PHASES-10-12.md for Assignment & Submission, Grading, and Real-time Chat services.

LMS Platform PRD - Phases 10-12

Core LMS Features - Part 2

Phase 10: Assignment & Submission Service

Phase Goals

1. Create assignment creation and management system
2. Implement student submission workflow with file uploads
3. Add deadline enforcement and late submission handling
4. Implement basic plagiarism detection
5. Support multiple assignment types (essay, coding, MCQ, file upload)

User Stories

ID	User Story	Priority	Acceptance Criteria
US10.1	As a **Teacher**, I want to create an assignment with instructions and deadline so that students know what to submit	Must-Have	- Create assignment with title, description, deadline, max marks - Attach reference files - Set submission type (file/text/code) - Publish to students
US10.2	As a **Student**, I want to submit my assignment before the deadline so that I can get graded	Must-Have	- Upload file (PDF/DOCX/ZIP) - Add submission comments - See submission

ID	User Story	Priority	Acceptance Criteria
			confirmation - Cannot submit after deadline (unless late allowed)
US10.3	As a **Student**, I want to see time remaining for assignment deadline so that I can plan my work	Must-Have	- Countdown timer shown - Email reminder 24 hours before deadline - Push notification 1 hour before deadline
US10.4	As a **Teacher**, I want to allow late submissions with penalty so that students have flexibility	Should-Have	- Enable late submission (up to X days) - Configure penalty (% per day) - Late submissions marked clearly - Penalty auto-applied to grade
US10.5	As a **Teacher**, I want basic plagiarism check on submissions so that I can detect copied work	Should-Have	- Run plagiarism check on text submissions - Similarity score shown - Highlight similar sections - Compare with previous submissions

Functional Requirements

FR10.1: Assignment CRUD

- Endpoints:
  - `POST /api/v1/assignments`: Create assignment

    - GET /api/v1/assignments: List assignments (filter by course, subject, status)
    - GET /api/v1/assignments/{id}: Get assignment details
    - PUT /api/v1/assignments/{id}: Update assignment
    - DELETE /api/v1/assignments/{id}: Delete assignment (if no submissions)
    - POST /api/v1/assignments/{id}/publish: Publish assignment
- Assignment fields:
    - title, description, courseId, subjectId, sectionId, createdBy
    - dueDate, maxMarks, submissionType (file/text/code/mcq), allowLate, latePenalty
    - attachments (S3 URLs), rubric, status (draft/published/closed)
- Validation: dueDate in future, maxMarks > 0

FR10.2: Submission Workflow

- Endpoints:
    - POST /api/v1/assignments/{id}/submit: Submit assignment
    - GET /api/v1/submissions/{id}: Get submission details
    - PUT /api/v1/submissions/{id}: Update submission (before deadline)
    - DELETE /api/v1/submissions/{id}: Withdraw submission
    - GET /api/v1/students/{id}/submissions: Get student's submissions
- Submission fields:
    - assignmentId, studentId, files (S3 URLs), textContent, submittedAt
    - isLate, lateDays, status (submitted/graded/returned)
- File upload: Max 50MB, types: PDF, DOCX, ZIP, JPG, PNG
- Validation: Student enrolled, not already submitted (unless resubmission allowed), within deadline or late window

FR10.3: Deadline Enforcement

- Check deadline on submission
- If past deadline and late not allowed: Reject with 400 error
- If late allowed: Accept but mark as late, calculate late days
- Scheduled job: Auto-close assignments 24 hours after late deadline
- Notifications:
    - 24 hours before: Email reminder
    - 1 hour before: Push notification
    - On deadline: Email to non-submitters

FR10.4: Late Submission Handling

- Late submission window: Configurable per assignment (e.g., 3 days)
- Penalty calculation: `penalty = lateDays * latePenaltyPerDay`
- Example: 2 days late, 5% penalty/day = 10% penalty
- Applied during grading: `finalMarks = (marks * (100 - penalty)) / 100`
- Late submissions highlighted in teacher's view

FR10.5: Plagiarism Detection (Basic)

- Text-based plagiarism check using string similarity algorithms
- Compare submission with:
  - Previous submissions in same assignment
  - Previous year submissions (if available)
  - Public sources (optional, via API like Copyleaks/Turnitin)
- Similarity score: 0-100%
- Threshold: >70% flagged for review
- Teacher can view similar sections side-by-side
- Note: Advanced plagiarism detection (Phase 25 - AI/ML)

FR10.6: Assignment Types

- **File Upload**: Student uploads file(s)
- **Text Submission**: Student types answer in text box
- **Code Submission**: Code editor with syntax highlighting, auto-test (optional)
- **MCQ**: Multiple choice questions, auto-graded (Phase 11)

FR10.7: Bulk Operations

- Endpoints:
  - `POST /api/v1/assignments/{id}/submissions/download`: Download all submissions as ZIP
  - `POST /api/v1/assignments/{id}/extend-deadline`: Extend deadline for all or specific students
- Use cases: Download all for offline grading, extend deadline due to technical issues

Non-Functional Requirements

NFR10.1: Performance

- Assignment creation: <500ms
- Submission upload: <3 seconds for 10MB file
- Plagiarism check: <10 seconds for 5000-word text
- Bulk download: <30 seconds for 100 submissions

NFR10.2: Reliability

- File upload resilience: Resume on failure
- Submission confirmation: Idempotent (prevent duplicate submissions)
- Deadline check: Server time (not client time)

NFR10.3: Storage

- S3 bucket: `lms-assignments-{env}`
- Folder structure: `{assignmentId}/{studentId}/{filename}`
- Retention: 2 years, then archive to Glacier

NFR10.4: Scalability

- Support 10,000 assignments
- 100,000 submissions
- 1000 concurrent submissions

Technical Architecture

New Service:

```
services/assignment-service/
├── Dockerfile
├── package.json
├── tsconfig.json
├── prisma/
│   ├── schema.prisma
│   └── migrations/
├── src/
```

```
│   ├── index.ts
│   ├── app.ts
│   ├── config/
│   │   ├── env.ts
│   │   ├── s3.ts
│   │   └── kafka.ts
│   ├── controllers/
│   │   ├── assignment.controller.ts
│   │   ├── submission.controller.ts
│   │   └── plagiarism.controller.ts
│   ├── services/
│   │   ├── assignment.service.ts
│   │   ├── submission.service.ts
│   │   ├── deadline.service.ts
│   │   ├── plagiarism.service.ts
│   │   ├── s3.service.ts
│   │   └── notification.service.ts
│   ├── utils/
│   │   ├── deadline-checker.ts
│   │   ├── penalty-calculator.ts
│   │   ├── similarity-checker.ts (Levenshtein, cosine similarity)
│   │   └── zip-generator.ts
│   ├── jobs/
│   │   ├── deadline-reminder.job.ts
│   │   ├── auto-close.job.ts
│   │   └── plagiarism-scan.job.ts
│   ├── middleware/
│   │   ├── enrollment-check.middleware.ts
│   │   └── file-upload.middleware.ts
│   └── routes/
│       ├── assignment.route.ts
│       └── submission.route.ts
└── tests/
    ├── unit/
    │   ├── deadline.service.test.ts
```

```
    |   └── plagiarism.service.test.ts
    └── integration/
        └── submission.test.ts
```

Data Models (Prisma):

```
model Assignment {
  id             String   @id @default(uuid())
  title          String
  description    String   @db.Text
  courseId       String
  subjectId      String
  sectionId      String?  // Null = all sections
  createdBy      String   // teacherId
  dueDate        DateTime
  maxMarks       Int
  submissionType String   // "file", "text", "code", "mcq"
  allowLate      Boolean  @default(false)
  lateDeadline   DateTime?
  latePenaltyPerDay Int?  // Percentage
  attachments    String[] // S3 URLs
  rubric         Json?
  status         String   @default("draft") // "draft", "published", "closed"
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt

  submissions    Submission[]

  @@index([courseId, dueDate])
  @@index([createdBy])
  @@index([status])
}

model Submission {
```

```
  id              String   @id @default(uuid())
  assignmentId    String
  studentId       String
  files           String[] // S3 URLs
  textContent     String?  @db.Text
  codeContent     String?  @db.Text
  submittedAt     DateTime @default(now())
  isLate          Boolean  @default(false)
  lateDays        Int?
  status          String   @default("submitted") // "submitted", "graded",
"returned"
  plagiarismScore Float?   // 0-100
  flaggedForReview Boolean @default(false)

  assignment      Assignment @relation(fields: [assignmentId], references: [id])
  grade           Grade?

  @@unique([assignmentId, studentId])
  @@index([studentId])
  @@index([assignmentId, submittedAt])
}

model PlagiarismCheck {
  id              String   @id @default(uuid())
  submissionId    String
  comparedWithId  String   // Another submission ID
  similarityScore Float    // 0-100
  matchedSections Json     // Array of { text, startIndex, endIndex }
  checkedAt       DateTime @default(now())

  @@index([submissionId])
  @@index([similarityScore])
}
```

Kafka Topics:

- `assignment.created`
- `assignment.published`
- `submission.created`
- `submission.late`
- `assignment.deadline.approaching`
- `plagiarism.detected`

S3 Bucket Structure:

```
lms-assignments-{env}/
├── {assignmentId}/
│   ├── attachments/
│   │   └── {filename}
│   └── submissions/
│       └── {studentId}/
│           └── {filename}
```

Dependencies & Tech Stack

- `@aws-sdk/client-s3`: S3 operations
- `multer-s3`: File upload to S3
- `archiver`: ZIP file generation
- `string-similarity`: Text similarity (or `natural` library)
- `bull`: Job scheduling
- `pdfkit`: Generate submission receipts (optional)

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V10.1	Create assignment	Postman	- Assignment created with all fields - Attachments uploaded to S3 - Status is “draft”

ID	Test Case	Tool	Acceptance Criteria
V10.2	Publish assignment	Postman	- Status changes to “published” - Kafka event published - Students notified
V10.3	Submit assignment	Postman	- Student uploads file - Submission saved - Confirmation email sent
V10.4	Deadline enforcement	Postman	- Submit after deadline (late not allowed) - Returns 400 error - Clear error message
V10.5	Late submission	Postman	- Submit 2 days late - Marked as late - lateDays = 2 - Penalty calculated
V10.6	Plagiarism detection	Jest	- Submit copied text - Similarity score >70% - Flagged for review - Teacher notified
V10.7	Deadline reminders	Manual (wait)	- 24 hours before: Email sent - 1 hour before: Push notification - On deadline: Non-submitters notified
V10.8	Bulk download	Postman	- Download all submissions - ZIP file contains all files - Organized by student
V10.9	Concurrent submissions	Artillery	- 100 students submit simultaneously - All submissions saved - No duplicates
V10.10	Large file upload	Postman	

ID	Test Case	Tool	Acceptance Criteria
			- Upload 50MB file - Upload completes - File accessible in S3

Phase 11: Grading & Assessment Engine

Phase Goals

1. Implement manual grading workflow for assignments
2. Create rubric-based grading system
3. Add auto-grading for MCQ assessments
4. Implement grade calculation (weighted average)
5. Generate grade distribution analytics and reports

User Stories

ID	User Story	Priority	Acceptance Criteria
US11.1	As a **Teacher**, I want to grade student submissions so that students receive feedback	Must-Have	- View submission - Enter marks (0 to maxMarks) - Add comments/ feedback - Save grade - Student notified
US11.2	As a **Teacher**, I want to use a rubric for consistent grading so that grading is fair	Should-Have	- Create rubric with criteria - Each criterion has points - Grade using rubric - Total auto-calculated

ID	User Story	Priority	Acceptance Criteria
US11.3	As a **Student**, I want to see my grade and feedback so that I can improve	Must-Have	- View grade after teacher releases - Read feedback comments - See rubric breakdown - Download graded submission
US11.4	As a **Teacher**, I want MCQ tests to be auto-graded so that I save time	Must-Have	- Create MCQ test with correct answers - Students take test - Auto-graded immediately - Results shown to student
US11.5	As a **Student**, I want to see my overall course grade so that I know my standing	Must-Have	- View weighted average grade - Breakdown by assignment/test - Grade letter (A/B/C/D/F) - Class rank (optional)

Functional Requirements

FR11.1: Manual Grading

- Endpoints:
  - `POST /api/v1/submissions/{id}/grade`: Grade submission
  - `PUT /api/v1/grades/{id}`: Update grade
  - `GET /api/v1/assignments/{id}/grades`: Get all grades for assignment
  - `POST /api/v1/grades/{id}/release`: Release grade to student

- Grade fields:
  - `submissionId, marks, maxMarks, feedback, gradedBy, gradedAt`
  - `rubricScores (JSON), status (draft/released)`
- Validation: marks <= maxMarks, marks >= 0
- Late penalty auto-applied: `finalMarks = marks - (marks * penalty / 100)`

FR11.2: Rubric-Based Grading

- Rubric structure:

```
{
  "criteria": [
    {
      "name": "Content Quality",
      "description": "Depth and accuracy of content",
      "maxPoints": 40,
      "levels": [
        { "name": "Excellent", "points": 40, "description": "..." },
        { "name": "Good", "points": 30, "description": "..." },
        { "name": "Fair", "points": 20, "description": "..." },
        { "name": "Poor", "points": 10, "description": "..." }
      ]
    },
    {
      "name": "Presentation",
      "maxPoints": 20,
      "levels": [...]
    }
  ]
}
```

- Endpoints:
  - `POST /api/v1/assignments/{id}/rubric`: Attach rubric
  - `GET /api/v1/rubrics/{id}`: Get rubric
- Grading with rubric: Teacher selects level for each criterion, total auto-calculated

FR11.3: Auto-Grading (MCQ)

- MCQ question structure:

```
{
  "question": "What is 2+2?",
  "options": ["2", "3", "4", "5"],
  "correctAnswer": 2, // Index of correct option
  "marks": 1
}
```

- Student answer: `{ questionId, selectedOption }`
- Auto-grading logic:
  - Compare selectedOption with correctAnswer
  - If match: Award full marks
  - If no match: 0 marks
  - Calculate total: Sum of all question marks
- Immediate result: Grade available as soon as test submitted
- Endpoints:
  - `POST /api/v1/tests`: Create MCQ test
  - `POST /api/v1/tests/{id}/submit`: Submit answers (auto-graded)
  - `GET /api/v1/tests/{id}/results`: Get results

FR11.4: Grade Calculation

- Course grade components:
  - Assignments: 40%
  - Mid-term: 20%
  - Final exam: 30%
  - Attendance: 10%
  - (Weights configurable per course)
- Calculation:

```
courseGrade = (assignmentAvg * 0.4) + (midterm * 0.2) + (final * 0.3) + (attendance * 0.1)
```

- Letter grade mapping:
  - A: 90-100
  - B: 80-89
  - C: 70-79
  - D: 60-69
  - F: <60
- Endpoints:
  - `GET /api/v1/students/{id}/course-grade/{courseId}`: Get overall grade
  - `POST /api/v1/courses/{id}/grade-config`: Configure grade weights

FR11.5: Grade Analytics

- Endpoints:
  - `GET /api/v1/assignments/{id}/analytics`: Assignment grade distribution
  - `GET /api/v1/courses/{id}/analytics`: Course grade analytics
- Metrics:
  - Average, median, mode
  - Standard deviation
  - Grade distribution (histogram)
  - Top performers, bottom performers
- Visualizations: Data for charts (frontend renders)

FR11.6: Grade Release & Notifications

- Grades initially in “draft” status (not visible to students)
- Teacher releases grades: `POST /api/v1/grades/release`
- Options: Release all, release individually, scheduled release
- Notification: Email + push notification on grade release
- Student can request regrade (optional workflow)

Non-Functional Requirements

NFR11.1: Performance

- Grade submission: <500ms
- Auto-grading (100 MCQs): <2 seconds
- Grade calculation: <1 second
- Analytics generation: <3 seconds for 1000 students

NFR11.2: Accuracy

- Auto-grading: 100% accuracy (deterministic)
- Grade calculation: Precise to 2 decimal places
- No rounding errors in weighted averages

NFR11.3: Fairness

- Rubric ensures consistent grading
- Blind grading option (hide student names)
- Audit trail: All grade changes logged

NFR11.4: Scalability

- Support 100,000 grades
- 1000 concurrent auto-grading operations

Technical Architecture

Assignment Service Updates:

```
services/assignment-service/
├── src/
│   ├── controllers/
│   │   ├── grading.controller.ts        [NEW]
│   │   ├── rubric.controller.ts         [NEW]
│   │   ├── mcq.controller.ts            [NEW]
│   │   └── analytics.controller.ts      [NEW]
│   ├── services/
│   │   ├── grading.service.ts           [NEW]
│   │   ├── rubric.service.ts            [NEW]
│   │   ├── auto-grading.service.ts      [NEW]
│   │   ├── grade-calculator.service.ts  [NEW]
│   │   └── analytics.service.ts         [NEW]
│   ├── utils/
│   │   ├── grade-calculator.ts          [NEW]
│   │   ├── letter-grade.ts              [NEW]
│   │   └── statistics.ts                [NEW]
```

```
|   └── routes/
|       ├── grading.route.ts          [NEW]
|       └── mcq.route.ts              [NEW]
```

Data Models:

```
model Grade {
  id           String   @id @default(uuid())
  submissionId String   @unique
  marks        Float
  maxMarks     Int
  finalMarks   Float    // After late penalty
  feedback     String?  @db.Text
  rubricScores Json?    // { criteriaId: points }
  gradedBy     String   // teacherId
  gradedAt     DateTime @default(now())
  status       String   @default("draft") // "draft", "released"
  releasedAt   DateTime?

  submission   Submission @relation(fields: [submissionId], references: [id])

  @@index([submissionId])
  @@index([status])
}

model Rubric {
  id          String   @id @default(uuid())
  name        String
  description String?
  criteria    Json     // Array of criteria objects
  createdBy   String
  createdAt   DateTime @default(now())

  @@index([createdBy])
```

```
}

model MCQTest {
  id          String   @id @default(uuid())
  title       String
  courseId    String
  subjectId   String
  duration    Int      // minutes
  totalMarks  Int
  questions   Json     // Array of question objects
  createdBy   String
  scheduledAt DateTime?
  dueDate     DateTime?
  status      String   @default("draft")
  createdAt   DateTime @default(now())

  attempts    MCQAttempt[]

  @@index([courseId])
}

model MCQAttempt {
  id          String   @id @default(uuid())
  testId      String
  studentId   String
  answers     Json     // Array of { questionId, selectedOption }
  score       Float
  maxScore    Int
  startedAt   DateTime @default(now())
  submittedAt DateTime?

  test        MCQTest  @relation(fields: [testId], references: [id])

  @@unique([testId, studentId])
  @@index([studentId])
```

```
}

model CourseGradeConfig {
  id              String   @id @default(uuid())
  courseId        String   @unique
  components      Json     // { "assignments": 40, "midterm": 20, ... }
  letterGradeMap  Json     // { "A": [90, 100], "B": [80, 89], ... }
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt
}
```

Kafka Topics:

- `grade.released`
- `test.completed`
- `grade.below.threshold` (trigger intervention)

Dependencies & Tech Stack

- `mathjs`: Statistical calculations
- `chart.js` (frontend): Grade distribution charts
- Existing: Prisma, Kafka, Redis

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V11.1	Manual grading	Postman	- Grade submission with marks and feedback - Late penalty applied - Grade saved
V11.2	Rubric grading	Postman	- Create rubric - Grade using rubric - Total auto-calculated - Matches expected

ID	Test Case	Tool	Acceptance Criteria
V11.3	Auto-grade MCQ	Postman	- Create test with 10 MCQs - Submit answers - Auto-graded instantly - Score correct
V11.4	Grade release	Postman	- Release grade - Status changes to “released” - Student notified - Grade visible to student
V11.5	Course grade calculation	Postman	- Get overall course grade - Weighted average correct - Letter grade correct
V11.6	Grade analytics	Postman	- Get assignment analytics - Average, median correct - Distribution data accurate
V11.7	Concurrent auto-grading	Artillery	- 100 students submit MCQ test simultaneously - All graded correctly - No errors
V11.8	Grade update audit	Database	- Update grade - Old grade logged - Audit trail complete

---

Phase 12: Real-time Chat Service

Phase Goals

1. Implement real-time messaging using Socket.IO
2. Support one-on-one and group chats (course-based)
3. Add file sharing in chat
4. Persist chat history in MongoDB

5. Implement typing indicators and read receipts

User Stories

ID	User Story	Priority	Acceptance Criteria
US12.1	As a **Student**, I want to chat with my teacher so that I can ask questions	Must-Have	- Send message to teacher - Receive reply in real-time - See message history - Notifications for new messages
US12.2	As a **Teacher**, I want a group chat for my course so that students can discuss	Must-Have	- Create course group chat - All enrolled students auto-added - Send messages to group - Messages visible to all
US12.3	As a **Student**, I want to share files in chat so that I can collaborate	Should-Have	- Upload file in chat (max 10MB) - File appears as message - Others can download - Supports images, PDFs, docs
US12.4	As a **Student**, I want to see when someone is typing so that I know they’re responding	Should-Have	- Typing indicator shown - Updates in real-time

ID	User Story	Priority	Acceptance Criteria
			- Disappears after 3 seconds of inactivity
US12.5	As a **Teacher**, I want to search chat history so that I can find past discussions	Should-Have	- Search by keyword - Filter by date, user - Results highlighted - Fast search (<2s)

Functional Requirements

FR12.1: Real-time Messaging (Socket.IO)

- Socket events:
  - `connection`: Client connects, authenticate via token
  - `join_room`: Join chat room (course, direct)
  - `send_message`: Send message to room
  - `receive_message`: Receive message from room
  - `typing`: User is typing
  - `stop_typing`: User stopped typing
  - `disconnect`: Client disconnects
- Message structure:

```
{
  "id": "uuid",
  "roomId": "course-123",
  "senderId": "user-456",
  "senderName": "John Doe",
  "senderType": "student",
  "content": "Hello, I have a question",
  "type": "text", // "text", "file", "image"
  "fileUrl": null,
  "timestamp": "2025-12-15T20:00:00Z",
```

```
  "readBy": ["user-789"]
}
```

- Authentication: Validate JWT token on connection
- Room types:
  - Direct: `direct-{userId1}-{userId2}` (sorted IDs)
  - Course: `course-{courseId}`
  - Section: `section-{sectionId}`

FR12.2: Chat Rooms

- Endpoints (REST for room management):
  - `POST /api/v1/chat/rooms`: Create room
  - `GET /api/v1/chat/rooms`: List user's rooms
  - `GET /api/v1/chat/rooms/{id}`: Get room details
  - `POST /api/v1/chat/rooms/{id}/members`: Add member
  - `DELETE /api/v1/chat/rooms/{id}/members/{userId}`: Remove member
- Room fields:
  - `id, name, type (direct/group/course), members (array of userIds), createdBy, createdAt`
- Auto-create course rooms: When course published, create room and add all enrolled students + teacher

FR12.3: Message Persistence (MongoDB)

- Store all messages in MongoDB for history
- Collections:
  - `rooms`: Room metadata
  - `messages`: All messages
  - `readReceipts`: Track who read which message
- Message schema (MongoDB):

```
{
  _id: ObjectId,
  roomId: String,
  senderId: String,
  senderName: String,
```

```
  senderType: String,
  content: String,
  type: String,
  fileUrl: String,
  timestamp: Date,
  readBy: [String], // Array of userIds
  editedAt: Date,
  deletedAt: Date
}
```

- Indexes: `roomId + timestamp`, `senderId`, `content (text index for search)`

FR12.4: File Sharing

- Upload flow:
  1. Client requests upload URL: `POST /api/v1/chat/upload`
  2. Server generates S3 presigned URL
  3. Client uploads to S3
  4. Client sends message with fileUrl via Socket
  5. Server saves message to MongoDB
- File types: Images (JPG, PNG), Documents (PDF, DOCX), Archives (ZIP)
- Size limit: 10MB
- S3 bucket: `lms-chat-files-{env}`
- Folder structure: `{roomId}/{messageId}/{filename}`

FR12.5: Typing Indicators

- Socket event: `typing` with `{ roomId, userId, userName }`
- Broadcast to room: “John is typing…”
- Auto-stop after 3 seconds of inactivity
- Multiple users typing: “John and 2 others are typing…”

FR12.6: Read Receipts

- Socket event: `mark_read` with `{ roomId, messageId }`
- Update message’s `readBy` array
- Show “Read by 5 people” or individual names (direct chat)
- Unread count: Count messages where current user not in `readBy`

FR12.7: Message Search

- Endpoint: `GET /api/v1/chat/rooms/{id}/search?q={query}&from={date}&to={date}`
- MongoDB text search on `content` field
- Filters: date range, sender
- Pagination: 20 results per page
- Highlight matched terms in results

FR12.8: Notifications

- New message notification:
  - If user offline: Push notification + email (configurable)
  - If user online but not in room: In-app notification
- Notification content: Sender name, message preview (first 50 chars)
- Mute room: User can mute notifications for specific room

Non-Functional Requirements

NFR12.1: Performance

- Message delivery latency: <100ms
- Message persistence: <50ms
- Search: <2 seconds for 100k messages
- Support 10,000 concurrent connections

NFR12.2: Scalability

- Horizontal scaling: Multiple Socket.IO instances with Redis adapter
- Redis pub/sub for cross-instance message delivery
- MongoDB sharding for large message volumes

NFR12.3: Reliability

- Message delivery guarantee: At-least-once (idempotency on client)
- Offline message queue: Store in Redis, deliver on reconnect
- Connection resilience: Auto-reconnect on disconnect

NFR12.4: Security

- Authenticate all Socket connections

- Validate room membership before sending/receiving
- Encrypt messages in transit (TLS)
- No XSS: Sanitize message content

Technical Architecture

New Service:

```
services/chat-service/
├── Dockerfile
├── package.json
├── tsconfig.json
├── src/
│   ├── index.ts
│   ├── app.ts
│   ├── config/
│   │   ├── env.ts
│   │   ├── socket.ts
│   │   ├── mongodb.ts
│   │   ├── redis.ts
│   │   └── s3.ts
│   ├── socket/
│   │   ├── index.ts
│   │   ├── auth.middleware.ts
│   │   ├── chat.handler.ts
│   │   ├── typing.handler.ts
│   │   └── room.handler.ts
│   ├── controllers/
│   │   ├── room.controller.ts
│   │   ├── message.controller.ts
│   │   └── upload.controller.ts
│   ├── services/
│   │   ├── room.service.ts
│   │   ├── message.service.ts
│   │   ├── search.service.ts
```

```
│   │   └── notification.service.ts
│   ├── models/ (MongoDB schemas)
│   │   ├── room.model.ts
│   │   ├── message.model.ts
│   │   └── readReceipt.model.ts
│   ├── utils/
│   │   ├── socket-auth.ts
│   │   └── message-sanitizer.ts
│   └── routes/
│       ├── room.route.ts
│       └── message.route.ts
└── tests/
    ├── unit/
    └── integration/
```

MongoDB Schemas (Mongoose):

```
// models/room.model.ts
const roomSchema = new Schema({
  _id: String, // Custom ID: "course-123" or "direct-user1-user2"
  name: String,
  type: { type: String, enum: ['direct', 'group', 'course'] },
  members: [{ userId: String, userType: String, joinedAt: Date }],
  createdBy: String,
  createdAt: { type: Date, default: Date.now },
  lastMessageAt: Date,
  metadata: Schema.Types.Mixed // Course info, etc.
});

// models/message.model.ts
const messageSchema = new Schema({
  roomId: { type: String, index: true },
  senderId: { type: String, index: true },
  senderName: String,
```

```
  senderType: String,
  content: { type: String, text: true }, // Text index for search
  type: { type: String, enum: ['text', 'file', 'image'] },
  fileUrl: String,
  fileName: String,
  fileSize: Number,
  timestamp: { type: Date, default: Date.now, index: true },
  readBy: [String],
  editedAt: Date,
  deletedAt: Date
});

messageSchema.index({ roomId: 1, timestamp: -1 });
```

Socket.IO Setup:

```
// socket/index.ts
import { Server } from 'socket.io';
import { createAdapter } from '@socket.io/redis-adapter';
import { createClient } from 'redis';

const io = new Server(server, {
  cors: { origin: process.env.FRONTEND_URL }
});

// Redis adapter for multi-instance scaling
const pubClient = createClient({ url: process.env.REDIS_URL });
const subClient = pubClient.duplicate();
io.adapter(createAdapter(pubClient, subClient));

// Authentication middleware
io.use(async (socket, next) => {
  const token = socket.handshake.auth.token;
  const user = await verifyToken(token);
```

```
  if (user) {
    socket.data.user = user;
    next();
  } else {
    next(new Error('Authentication failed'));
  }
});

// Event handlers
io.on('connection', (socket) => {
  console.log(`User connected: ${socket.data.user.id}`);

  socket.on('join_room', (roomId) => {
    // Verify membership, then join
    socket.join(roomId);
  });

  socket.on('send_message', async (data) => {
    // Save to MongoDB
    const message = await saveMessage(data);
    // Broadcast to room
    io.to(data.roomId).emit('receive_message', message);
  });

  socket.on('typing', (data) => {
    socket.to(data.roomId).emit('user_typing', {
      userId: socket.data.user.id,
      userName: socket.data.user.name
    });
  });

  socket.on('disconnect', () => {
    console.log(`User disconnected: ${socket.data.user.id}`);
```

```
  });
});
```

Kong Gateway Update:

```
services:
  - name: chat-service
    url: http://chat-service:3003
    routes:
      - name: chat-api
        paths: ["/api/v1/chat"]
      - name: chat-socket
        paths: ["/socket.io"]
        strip_path: false
```

Kafka Topics:

- `chat.message.sent`
- `chat.room.created`
- `chat.file.uploaded`

Dependencies & Tech Stack

- `socket.io`: Real-time communication
- `@socket.io/redis-adapter`: Multi-instance scaling
- `mongoose`: MongoDB ODM
- `ioredis`: Redis client
- `@aws-sdk/client-s3`: File uploads
- `sanitize-html`: XSS prevention

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V12.1	Real-time messaging	Socket.IO client	

ID	Test Case	Tool	Acceptance Criteria
			- User A sends message - User B receives instantly - Latency <100ms
V12.2	Create course chat room	Postman	- Create room - Members auto-added - Room visible in list
V12.3	File sharing	Socket.IO client	- Upload image in chat - File saved to S3 - Others can download
V12.4	Typing indicator	Socket.IO client	- User A types - User B sees “A is typing” - Disappears after 3s
V12.5	Read receipts	Socket.IO client	- User B reads message - User A sees “Read by B” - Unread count updates
V12.6	Message search	Postman	- Search for keyword - Returns matching messages - Response time <2s
V12.7	Offline message delivery	Manual	- User A offline - User B sends message - User A reconnects - Message delivered
V12.8	Concurrent connections	Socket.IO load test	- 1000 users connect - All receive messages - No dropped connections
V12.9	Cross-instance messaging	Manual (2 servers)	- User A on server 1 - User B on server 2 - Messages delivered via Redis

ID	Test Case	Tool	Acceptance Criteria
V12.10	Message persistence	Database check	- Send 100 messages - All saved to MongoDB - Retrievable via API

End of Phases 10-12

Next Document: 04-PHASES-13-18.md for Search, Analytics, Notifications, Timetable, Library, and Payment services.

LMS Platform PRD - Phases 13-18

Advanced Features & Integrations

Phase 13: Search Service (Elasticsearch)

Phase Goals

1. Implement full-text search across courses, users, content, and messages
2. Set up Elasticsearch indexing pipeline via Kafka
3. Add advanced filters and faceted search
4. Implement autocomplete/suggestions
5. Track search analytics

User Stories

ID	User Story	Priority	Acceptance Criteria
US 13.1	As a **Student**, I want to search for courses by keyword so that I can find relevant courses quickly	Must-Have	- Search returns relevant courses - Results ranked by relevance - Response time <1 second - Highlights matched terms
US 13.2	As a **Teacher**, I want to search for students by name or ID so that I can find student records	Must-Have	- Search across name, email, registration number - Fuzzy matching for

ID	User Story	Priority	Acceptance Criteria
			typos - Filter by batch, section - Export results
US 13.3	As a **Student**, I want autocomplete suggestions while typing so that I can search faster	Should-Have	- Suggestions appear after 3 characters - Shows top 5 matches - Updates in real-time - Keyboard navigation

Functional Requirements

FR13.1: Elasticsearch Setup

- Indices: `courses`, `users`, `content`, `messages`
- Index mappings with analyzers for text fields
- Synonym support (e.g., “CS” → “Computer Science”)
- Custom scoring for relevance

FR13.2: Indexing Pipeline

- Kafka consumers listen to entity creation/update events
- Transform data and index to Elasticsearch
- Bulk indexing for initial data load
- Endpoints:
  - `POST /api/v1/search/reindex`: Trigger full reindex
  - `GET /api/v1/search/index-status`: Check indexing progress

FR13.3: Search API

- Endpoints:
  - `GET /api/v1/search?q={query}&type={type}&filters={...}`: Universal search
  - `GET /api/v1/search/courses?q={query}`: Course-specific search
  - `GET /api/v1/search/suggest?q={query}`: Autocomplete

- Filters: type, date range, branch, status, tags
- Pagination: 20 results per page
- Facets: Count by type, branch, etc.

FR13.4: Search Analytics

- Track: search queries, result clicks, zero-result queries
- Store in database for analysis
- Dashboard: Popular searches, failed searches, click-through rate

Technical Architecture

```
services/search-service/
├── src/
│   ├── controllers/
│   │   ├── search.controller.ts
│   │   └── index.controller.ts
│   ├── services/
│   │   ├── elasticsearch.service.ts
│   │   ├── indexer.service.ts
│   │   └── analytics.service.ts
│   ├── workers/
│   │   ├── course-indexer.worker.ts
│   │   ├── user-indexer.worker.ts
│   │   └── content-indexer.worker.ts
│   └── config/
│       └── elasticsearch.config.ts
```

Data Model (Elasticsearch):

```
// courses index
{
  "mappings": {
    "properties": {
      "id": { "type": "keyword" },
```

```
      "name": { "type": "text", "analyzer": "standard" },
      "code": { "type": "keyword" },
      "description": { "type": "text" },
      "branch": { "type": "keyword" },
      "credits": { "type": "integer" },
      "tags": { "type": "keyword" },
      "createdAt": { "type": "date" }
    }
  }
}
```

Verification Steps

ID	Test	Acceptance
V13.1	Search courses	Returns relevant results, <1s response
V13.2	Fuzzy search	Finds “Jhon” when searching “John”
V13.3	Autocomplete	Shows suggestions after 3 chars
V13.4	Filters	Combine multiple filters correctly
V13.5	Bulk indexing	Index 10k records in <5 minutes

Phase 14: Analytics & Reporting Service

Phase Goals

1. Build analytics dashboards for students, teachers, and admins
2. Generate pre-built reports (attendance, grades, enrollment)
3. Create custom report builder
4. Implement data export (CSV, PDF, Excel)
5. Add scheduled report delivery

User Stories

ID	User Story	Priority	Acceptance Criteria
US 14.1	As a **Student**, I want to see my performance dashboard so that I can track my progress	Must-Have	- Shows GPA, attendance %, assignment completion - Graphs for trends - Comparison with class average - Exportable as PDF
US 14.2	As an **HOD**, I want to generate department-wide reports so that I can analyze performance	Must-Have	- Select report type, filters, date range - Preview before export - Export to CSV/PDF/ Excel - Schedule weekly delivery
US 14.3	As a **Dean**, I want custom reports with specific metrics so that I can make data-driven decisions	Should-Have	- Drag-and-drop report builder - Select dimensions and metrics - Save report templates - Share with stakeholders

Functional Requirements

FR14.1: Student Analytics

- Metrics: GPA, attendance %, assignment completion rate, course progress
- Visualizations: Line charts (trends), bar charts (comparison), pie charts (distribution)
- Endpoint: GET /api/v1/analytics/students/{id}/dashboard

FR14.2: Teacher Analytics

- Metrics: Class average, attendance rate, assignment submission rate, grade distribution
- Endpoint: `GET /api/v1/analytics/teachers/{id}/dashboard`

FR14.3: Admin Analytics

- Metrics: Enrollment trends, course popularity, resource utilization, student retention
- Endpoint: `GET /api/v1/analytics/admin/dashboard`

FR14.4: Pre-built Reports

- Report types:
  - Attendance Report: By course, section, date range
  - Grade Report: By course, student, assignment
  - Enrollment Report: By semester, branch, course
  - Defaulter Report: Students below attendance/grade threshold
- Endpoint: `POST /api/v1/reports/generate`

FR14.5: Custom Report Builder

- Dimensions: Student, Course, Branch, Date, Teacher
- Metrics: Count, Average, Sum, Min, Max
- Filters: Date range, status, type
- Endpoint: `POST /api/v1/reports/custom`

FR14.6: Export & Scheduling

- Export formats: CSV, PDF, Excel
- Scheduled reports: Daily, weekly, monthly
- Email delivery to recipients
- Endpoint: `POST /api/v1/reports/schedule`

Technical Architecture

```
services/analytics-service/
├── src/
│   ├── controllers/
│   │   ├── analytics.controller.ts
```

```
│   │   ├── report.controller.ts
│   │   └── export.controller.ts
│   ├── services/
│   │   ├── student-analytics.service.ts
│   │   ├── teacher-analytics.service.ts
│   │   ├── admin-analytics.service.ts
│   │   ├── report-generator.service.ts
│   │   └── export.service.ts
│   ├── utils/
│   │   ├── csv-generator.ts
│   │   ├── pdf-generator.ts
│   │   └── excel-generator.ts
│   ├── jobs/
│   │   └── scheduled-reports.job.ts
│   └── templates/
│       └── report-templates/
```

Dependencies:

- `exceljs`: Excel generation
- `pdfkit`: PDF generation
- `csv-writer`: CSV generation
- `chart.js`: Chart generation (server-side)

Verification Steps

ID	Test	Acceptance
V14.1	Student dashboard	Shows all metrics, charts render
V14.2	Generate attendance report	Correct data, exports to CSV
V14.3	Custom report	Build report, save template, export
V14.4	Scheduled report	Runs on schedule, emails sent
V14.5	Large dataset export	Export 10k records in <30s

Phase 15: Notification Enhancement (SMS & Push)

Phase Goals

1. Integrate SMS notifications via Twilio
2. Implement web push notifications
3. Add mobile push notifications (FCM)
4. Create notification preferences management
5. Build notification history and tracking

User Stories

ID	User Story	Priority	Acceptance Criteria
US 15.1	As a **Student**, I want SMS notifications for important events so that I don't miss deadlines	Must-Have	- Receive SMS for assignment deadlines, grade releases - SMS delivered within 1 minute - Can opt-out per notification type
US 15.2	As a **Teacher**, I want push notifications on my phone so that I can respond quickly	Must-Have	- Push sent for new messages, submissions - Notification shows preview - Tapping opens relevant page
US 15.3	As a **Student**, I want to manage notification preferences so that I control what I receive	Should-Have	- Enable/disable per channel (email, SMS, push) - Enable/disable per event

ID	User Story	Priority	Acceptance Criteria
			type - Quiet hours setting

Functional Requirements

FR15.1: SMS Integration (Twilio)

- Endpoints:
  - `POST /api/v1/notifications/sms/send`: Send SMS
- SMS templates: Assignment due, grade released, attendance low, fee due
- Rate limiting: Max 100 SMS/day per user
- Cost tracking: Log SMS count for billing

FR15.2: Web Push Notifications

- Use Web Push API (service workers)
- Endpoints:
  - `POST /api/v1/notifications/push/subscribe`: Subscribe to push
  - `POST /api/v1/notifications/push/send`: Send push notification
- Notification payload: title, body, icon, data (deep link)
- Browser support: Chrome, Firefox, Edge, Safari

FR15.3: Mobile Push (FCM)

- Firebase Cloud Messaging integration
- Endpoints:
  - `POST /api/v1/notifications/fcm/register`: Register device token
  - `POST /api/v1/notifications/fcm/send`: Send to device
- Notification types: Data messages, notification messages
- Topic subscriptions: Course-based, role-based

FR15.4: Notification Preferences

- Endpoints:
  - `GET /api/v1/users/{id}/notification-preferences`: Get preferences
  - `PUT /api/v1/users/{id}/notification-preferences`: Update preferences
- Preferences structure:

```
{
  "email": { "assignmentDue": true, "gradeReleased": true },
  "sms": { "assignmentDue": true, "gradeReleased": false },
  "push": { "newMessage": true, "assignmentDue": true },
  "quietHours": { "start": "22:00", "end": "08:00" }
}
```

FR15.5: Notification History

- Endpoints:
  - `GET /api/v1/users/{id}/notifications`: Get notification history
  - `PUT /api/v1/notifications/{id}/read`: Mark as read
- Store: notification type, channel, status (sent/failed), timestamp
- Retention: 90 days

Technical Architecture

```
services/notification-service/ (UPDATE existing)
├── src/
│   ├── services/
│   │   ├── sms.service.ts          [NEW]
│   │   ├── web-push.service.ts     [NEW]
│   │   ├── fcm.service.ts          [NEW]
│   │   └── preference.service.ts   [NEW]
│   ├── handlers/
│   │   ├── sms.handler.ts          [NEW]
│   │   └── push.handler.ts         [NEW]
│   └── utils/
│       └── quiet-hours-checker.ts  [NEW]
```

Data Model:

```
model NotificationPreference {
  id          String   @id @default(uuid())
```

```
  userId      String   @unique
  userType    String
  channels    Json     // { email: {...}, sms: {...}, push: {...} }
  quietHours  Json?    // { start, end }
  updatedAt   DateTime @updatedAt
}

model NotificationLog {
  id          String   @id @default(uuid())
  userId      String
  type        String   // "assignment_due", "grade_released"
  channel     String   // "email", "sms", "push"
  status      String   // "sent", "failed", "delivered"
  sentAt      DateTime @default(now())
  deliveredAt DateTime?
  error       String?

  @@index([userId, sentAt])
}

model DeviceToken {
  id        String   @id @default(uuid())
  userId    String
  token     String   @unique
  platform  String   // "web", "ios", "android"
  createdAt DateTime @default(now())

  @@index([userId])
}
```

Dependencies:

- `twilio`: SMS service
- `web-push`: Web push notifications
- `firebase-admin`: FCM

Verification Steps

ID	Test	Acceptance
V15.1	Send SMS	SMS delivered within 1 minute
V15.2	Web push	Push received in browser
V15.3	Mobile push	Push received on mobile app
V15.4	Preferences	Update preferences, notifications respect settings
V15.5	Quiet hours	No notifications during quiet hours

Phase 16: Timetable & Scheduling Service

Phase Goals

1. Create timetable management system
2. Implement room/resource booking
3. Add conflict detection algorithm
4. Generate iCal exports for calendar integration
5. Support recurring events and exceptions

User Stories

ID	User Story	Priority	Acceptance Criteria
US16.1	As an **HOD**, I want to create a timetable for my department so that classes are scheduled	Must-Have	- Add classes with time, room, teacher - Detect conflicts (room, teacher) - Publish to students - Export to PDF

ID	User Story	Priority	Acceptance Criteria
US16.2	As a **Student**, I want to view my timetable so that I know when and where my classes are	Must-Have	- See weekly view - Color-coded by subject - Shows room, teacher - Sync to Google Calendar
US16.3	As a **Teacher**, I want to book a room for extra class so that I have a space	Should-Have	- Check room availability - Book for specific time - Receive confirmation - Cancel booking

Functional Requirements

FR16.1: Timetable CRUD

- Endpoints:
  - `POST /api/v1/timetables`: Create timetable
  - `GET /api/v1/timetables/{id}`: Get timetable
  - `POST /api/v1/timetables/{id}/slots`: Add time slot
  - `PUT /api/v1/timetable-slots/{id}`: Update slot
  - `DELETE /api/v1/timetable-slots/{id}`: Delete slot
- Slot fields: day, startTime, endTime, subjectId, teacherId, roomId, sectionId, type (lecture/lab)

FR16.2: Conflict Detection

- Check conflicts before saving:
  - Room conflict: Same room, overlapping time
  - Teacher conflict: Same teacher, overlapping time
  - Student conflict: Same section, overlapping time

- Return detailed conflict information
- Allow override with reason (admin only)

FR16.3: Room Booking

- Endpoints:
  - `GET /api/v1/rooms/availability?date={date}&startTime={time}&duration={mins}`: Check availability
  - `POST /api/v1/bookings`: Book room
  - `DELETE /api/v1/bookings/{id}`: Cancel booking
- Booking types: Regular class, extra class, exam, event
- Approval workflow: Teacher requests, admin approves

FR16.4: Calendar Integration

- Endpoints:
  - `GET /api/v1/timetables/{id}/ical`: Export as iCal
  - `GET /api/v1/students/{id}/timetable/ical`: Student's timetable
- iCal format with recurring events
- Subscribe URL for auto-sync

FR16.5: Recurring Events

- Support weekly recurring (e.g., every Monday 10:00-11:00)
- Exception handling (skip specific dates, e.g., holidays)
- Bulk operations (create semester timetable in one go)

Technical Architecture

```
services/timetable-service/
├── src/
│   ├── controllers/
│   │   ├── timetable.controller.ts
│   │   ├── slot.controller.ts
│   │   ├── room.controller.ts
│   │   └── booking.controller.ts
│   ├── services/
│   │   ├── timetable.service.ts
```

```
|   |   ├── conflict-detector.service.ts
|   |   ├── booking.service.ts
|   |   └── ical.service.ts
|   ├── utils/
|   |   ├── time-overlap-checker.ts
|   |   └── ical-generator.ts
|   └── algorithms/
|       └── auto-scheduler.ts (optional)
```

Data Model:

```
model Timetable {
  id           String   @id @default(uuid())
  name         String
  academicYear String
  semester     Int
  branchId     String
  createdBy    String
  status       String   @default("draft")
  createdAt    DateTime @default(now())

  slots        TimetableSlot[]
}

model TimetableSlot {
  id          String   @id @default(uuid())
  timetableId String
  day         Int      // 1=Monday, 7=Sunday
  startTime   String   // "10:00"
  endTime     String   // "11:00"
  subjectId   String
  teacherId   String
  roomId      String
  sectionId   String
```

```
  type        String   // "lecture", "lab", "tutorial"
  isRecurring Boolean  @default(true)
  exceptions  Json?    // Array of dates to skip

  timetable   Timetable @relation(fields: [timetableId], references: [id])

  @@index([timetableId, day])
}

model Room {
  id       String  @id @default(uuid())
  name     String
  building String
  capacity Int
  type     String  // "classroom", "lab", "auditorium"
  facilities String[] // "projector", "ac", "computers"
}

model RoomBooking {
  id        String   @id @default(uuid())
  roomId    String
  bookedBy  String
  purpose   String
  date      DateTime
  startTime String
  endTime   String
  status    String   @default("pending") // "pending", "approved", "rejected"
  createdAt DateTime @default(now())

  @@index([roomId, date])
}
```

Dependencies:

- `ical-generator`: iCal file generation

- `moment`: Time manipulation

Verification Steps

ID	Test	Acceptance
V16.1	Create timetable	Timetable created with slots
V16.2	Conflict detection	Room conflict detected, save rejected
V16.3	Room booking	Book room, availability updated
V16.4	iCal export	Export works, imports to Google Calendar
V16.5	Recurring events	Weekly classes created correctly

Phase 17: Library Management Integration

Phase Goals

1. Build library catalog system
2. Implement book issue/return workflow
3. Add fine calculation for late returns
4. Integrate with existing library systems (optional)
5. Implement book search and reservation

User Stories

ID	User Story	Priority	Acceptance Criteria
US17.1	As a **Student**, I want to search for books in the library so that I can find resources	Must-Have	- Search by title, author, ISBN - See availability status - Reserve if

ID	User Story	Priority	Acceptance Criteria
			available - View due date if issued
US17.2	As a **Librarian**, I want to issue books to students so that I can track borrowing	Must-Have	- Scan student ID and book barcode - Issue book with due date - Student notified - Transaction recorded
US17.3	As a **Student**, I want to see my borrowed books and fines so that I can return on time	Must-Have	- List of issued books - Due dates highlighted - Fine amount if overdue - Payment link

Functional Requirements

FR17.1: Book Catalog

- Endpoints:
  - `POST /api/v1/library/books`: Add book
  - `GET /api/v1/library/books`: Search books
  - `GET /api/v1/library/books/{id}`: Get book details
  - `PUT /api/v1/library/books/{id}`: Update book
- Book fields: title, author, ISBN, publisher, category, totalCopies, availableCopies, location

FR17.2: Issue/Return Workflow

- Endpoints:
  - `POST /api/v1/library/issue`: Issue book

    - `POST /api/v1/library/return`: Return book
    - `GET /api/v1/library/transactions`: Get transactions
- Issue validation: Book available, student eligible (no pending fines >₹500)
- Due date: 14 days from issue
- Auto-notification: 2 days before due date

FR17.3: Fine Calculation

- Fine rate: ₹5 per day per book
- Grace period: 1 day
- Max fine: ₹500 per book
- Calculation: `fine = max(0, min((daysLate - gracePeriod) * ratePerDay, maxFine))`
- Endpoint: `GET /api/v1/library/students/{id}/fines`

FR17.4: Reservation System

- Endpoints:
    - `POST /api/v1/library/reserve`: Reserve book
    - `GET /api/v1/library/reservations`: Get reservations
    - `DELETE /api/v1/library/reservations/{id}`: Cancel reservation
- Queue-based: First reserved, first issued when available
- Notification when book available (48-hour claim window)

FR17.5: Integration (Optional)

- REST API for external library systems (Koha, Evergreen)
- Sync catalog, transactions
- Webhook for real-time updates

Technical Architecture

```
services/library-service/
├── src/
│   ├── controllers/
│   │   ├── catalog.controller.ts
│   │   ├── transaction.controller.ts
│   │   ├── fine.controller.ts
```

```
|   |   └── reservation.controller.ts
|   ├── services/
|   |   ├── catalog.service.ts
|   |   ├── transaction.service.ts
|   |   ├── fine.service.ts
|   |   └── integration.service.ts
|   ├── jobs/
|   |   ├── due-date-reminder.job.ts
|   |   └── fine-calculator.job.ts
|   └── integrations/
|       └── koha-api.ts (optional)
```

Data Model:

```
model Book {
  id            String   @id @default(uuid())
  isbn          String   @unique
  title         String
  author        String
  publisher     String?
  category      String
  totalCopies   Int
  availableCopies Int
  location      String
  createdAt     DateTime @default(now())

  transactions  BookTransaction[]
  reservations  BookReservation[]
}

model BookTransaction {
  id        String   @id @default(uuid())
  bookId    String
  studentId String
```

```
  issuedAt   DateTime @default(now())
  dueDate    DateTime
  returnedAt DateTime?
  fine       Float    @default(0)
  status     String   // "issued", "returned", "overdue"

  book       Book     @relation(fields: [bookId], references: [id])

  @@index([studentId, status])
}

model BookReservation {
  id         String   @id @default(uuid())
  bookId     String
  studentId  String
  reservedAt DateTime @default(now())
  expiresAt  DateTime?
  status     String   @default("pending") // "pending", "fulfilled", "cancelled"

  book       Book     @relation(fields: [bookId], references: [id])

  @@index([bookId, status])
}
```

Verification Steps

ID	Test	Acceptance
V17.1	Search books	Returns relevant books, shows availability
V17.2	Issue book	Book issued, availableCopies decremented
V17.3	Return book	Book returned, fine calculated correctly
V17.4	Reserve book	Reservation created, notified when available

ID	Test	Acceptance
V17.5	Fine calculation	Overdue by 5 days = ₹20 fine (₹5/day, 1 day grace)

Phase 18: Payment & Fee Management

Phase Goals

1. Create fee structure management
2. Integrate payment gateway (Stripe/Razorpay)
3. Generate receipts and invoices
4. Track payment history
5. Implement refund workflow

User Stories

ID	User Story	Priority	Acceptance Criteria
US 18.1	As a **Student**, I want to pay my fees online so that I don’t have to visit the office	Must-Have	- View fee breakdown - Pay via card/UPI/ net banking - Receive receipt immediately - Payment confirmation email
US 18.2	As a **Finance Admin**, I want to configure fee structures so that different courses have different fees	Must-Have	- Create fee components (tuition, lab, library) - Assign to courses/ batches - Set due dates - Apply discounts/ scholarships

ID	User Story	Priority	Acceptance Criteria
US 18.3	As a **Student**, I want to see my payment history so that I can track my transactions	Must-Have	- List all payments - Download receipts - See pending dues - Payment reminders

Functional Requirements

FR18.1: Fee Structure

- Endpoints:
  - `POST /api/v1/fees/structures`: Create fee structure
  - `GET /api/v1/fees/structures`: List structures
  - `POST /api/v1/fees/assign`: Assign to students
- Components: Tuition, lab fee, library fee, hostel fee, exam fee
- Installments: Semester-wise, monthly
- Discounts: Scholarship, merit-based, sibling

FR18.2: Payment Gateway Integration

- Gateways: Razorpay (India), Stripe (International)
- Endpoints:
  - `POST /api/v1/payments/initiate`: Create payment order
  - `POST /api/v1/payments/verify`: Verify payment
  - `GET /api/v1/payments/{id}`: Get payment status
- Flow:
  1. Student initiates payment
  2. Create order in gateway
  3. Redirect to gateway
  4. Gateway callback on success/failure
  5. Verify signature
  6. Update payment status
  7. Generate receipt

FR18.3: Receipt Generation

- Auto-generate on successful payment

- PDF format with college letterhead
- Details: Student info, fee breakdown, amount paid, transaction ID, date
- Store in S3, send via email
- Endpoint: `GET /api/v1/payments/{id}/receipt`

FR18.4: Payment History

- Endpoints:
  - `GET /api/v1/students/{id}/payments`: Payment history
  - `GET /api/v1/students/{id}/dues`: Pending dues
- Show: Amount, date, status, receipt link
- Filter by academic year, semester

FR18.5: Refund Workflow

- Endpoints:
  - `POST /api/v1/payments/{id}/refund`: Initiate refund
  - `GET /api/v1/refunds/{id}`: Refund status
- Approval: Finance admin approves
- Process via gateway
- Notification on completion

Technical Architecture

```
services/payment-service/
├── src/
│   ├── controllers/
│   │   ├── fee.controller.ts
│   │   ├── payment.controller.ts
│   │   ├── receipt.controller.ts
│   │   └── refund.controller.ts
│   ├── services/
│   │   ├── fee.service.ts
│   │   ├── payment-gateway.service.ts
│   │   ├── receipt.service.ts
│   │   └── refund.service.ts
│   ├── integrations/
```

```
|   |   ├── razorpay.ts
|   |   └── stripe.ts
|   └── utils/
|       ├── signature-verifier.ts
|       └── pdf-generator.ts
```

Data Model:

```
model FeeStructure {
  id          String   @id @default(uuid())
  name        String
  courseId    String?
  batchId     String?
  components  Json     // [{ name, amount, type }]
  totalAmount Float
  dueDate     DateTime
  createdAt   DateTime @default(now())
}

model Payment {
  id           String   @id @default(uuid())
  studentId    String
  feeStructureId String
  amount       Float
  gateway      String   // "razorpay", "stripe"
  orderId      String   @unique
  transactionId String?
  status       String   // "pending", "success", "failed"
  paidAt       DateTime?
  receiptUrl   String?
  createdAt    DateTime @default(now())

  @@index([studentId])
}
```

```
model Refund {
  id          String   @id @default(uuid())
  paymentId   String
  amount      Float
  reason      String
  status      String   @default("pending")
  approvedBy  String?
  processedAt DateTime?
  createdAt   DateTime @default(now())
}
```

Dependencies:

- `razorpay`: Razorpay SDK
- `stripe`: Stripe SDK
- `pdfkit`: Receipt generation

Verification Steps

ID	Test	Acceptance
V18.1	Create fee structure	Structure created, assigned to students
V18.2	Payment flow	Initiate → Pay → Verify → Receipt generated
V18.3	Receipt generation	PDF generated, contains all details
V18.4	Payment history	Shows all payments, correct amounts
V18.5	Refund	Refund initiated, processed, amount credited

End of Phases 13-18

Next Document: `05-PHASES-19-25.md` for Frontend Portals, AWS Infrastructure, CI/CD, Mobile Apps, and AI/ML features.

LMS Platform PRD - Phases 19-20

Frontend Applications - Admin & College Portals

Phase 19: Super Admin Portal (Next.js)

Phase Goals

1. Build Next.js application for organization and super admin users
2. Implement organization dashboard with key metrics
3. Create college management interface (CRUD operations)
4. Add user management for all roles across colleges
5. Implement system settings and configuration UI
6. Build responsive design for desktop and tablet

User Stories

ID	User Story	Priority	Acceptance Criteria
US 19.1	As an **Organization Admin**, I want to see a dashboard with key metrics so that I can monitor the platform	Must-Have	- Shows total colleges, users, courses - Active users graph (daily/weekly/monthly) - Recent activities feed - Quick actions (add college, view reports)
		Must-Have	- Create college with details (name, address, contact)

ID	User Story	Priority	Acceptance Criteria
US 19.2	As an **Organization Admin**, I want to create and manage colleges so that I can onboard new institutions		- Upload college logo - Assign college admin - Edit/deactivate colleges - View college details
US 19.3	As an **Organization Admin**, I want to manage users across all colleges so that I can handle support requests	Must-Have	- Search users by name, email, role, college - View user details - Reset passwords - Deactivate accounts - View user activity logs
US 19.4	As an **Organization Admin**, I want to configure system settings so that I can customize the platform	Should-Have	- Set platform name, logo, theme colors - Configure email templates - Set default policies (password, session) - Manage API keys - Configure integrations
US 19.5	As an **Organization Admin**, I want to view analytics across all colleges so that I can identify trends	Should-Have	- Multi-college comparison charts - Enrollment trends - User growth - System usage metrics - Export reports

Functional Requirements

FR19.1: Dashboard

- Key metrics cards:
  - Total Colleges (with growth %)
  - Total Users (breakdown by role)
  - Active Courses
  - System Health (uptime, API response time)
- Charts:
  - User growth (line chart, last 30 days)
  - College-wise user distribution (bar chart)
  - Top active colleges (table)
- Recent activities feed (last 50 activities)
- Quick actions: Add College, View Reports, System Settings

FR19.2: College Management

- Pages:
  - `/colleges` - List all colleges (table with search, filter, pagination)
  - `/colleges/new` - Create college form
  - `/colleges/{id}` - College details page
  - `/colleges/{id}/edit` - Edit college form
- College form fields:
  - Basic: Name, code, address, city, state, country, pincode
  - Contact: Phone, email, website
  - Admin: College admin user (create or select existing)
  - Settings: Logo upload, timezone, academic year start
- Actions: Create, Edit, Deactivate, View Details, Assign Admin
- Validation: Unique college code, valid email/phone, required fields

FR19.3: User Management

- Pages:
  - `/users` - List all users (table with advanced filters)
  - `/users/{id}` - User details page
  - `/users/{id}/edit` - Edit user
- Filters: Role, college, status (active/inactive), date joined
- Search: Name, email, registration number

- User details:
    - Profile information
    - Role and permissions
    - Associated college/branch/section
    - Activity logs (last 100 actions)
    - Login history
- Actions: Reset Password, Deactivate, Send Email, View Activity

FR19.4: System Settings

- Pages:
    - `/settings/general` - Platform name, logo, theme
    - `/settings/email` - Email templates, SMTP config
    - `/settings/security` - Password policy, session timeout
    - `/settings/integrations` - API keys, webhooks
    - `/settings/notifications` - Notification preferences
- General settings:
    - Platform name, tagline
    - Logo upload (light/dark mode)
    - Primary/secondary colors
    - Favicon
- Email settings:
    - SMTP configuration
    - Email templates (welcome, password reset, etc.)
    - Preview and test email
- Security settings:
    - Password policy (length, complexity, expiry)
    - Session timeout
    - 2FA enforcement
    - IP whitelist

FR19.5: Analytics & Reports

- Pages:
    - `/analytics/overview` - Platform-wide analytics
    - `/analytics/colleges` - College comparison
    - `/analytics/users` - User analytics
    - `/reports` - Pre-built and custom reports

- Metrics:
    - User growth (daily, weekly, monthly)
    - Active users (DAU, WAU, MAU)
    - Course enrollment trends
    - System usage (API calls, storage)
- Visualizations: Line charts, bar charts, pie charts, tables
- Export: CSV, PDF, Excel
- Date range selector, college filter

FR19.6: Audit Logs

- Page: `/audit-logs`
- Show all admin actions:
    - User created/updated/deleted
    - College created/updated
    - Settings changed
    - Reports generated
- Filters: Action type, user, date range
- Export logs

Non-Functional Requirements

NFR19.1: Performance

- Initial page load: <2 seconds
- Dashboard metrics load: <1 second
- Table pagination: <500ms
- Search results: <300ms

NFR19.2: Responsiveness

- Desktop: 1920x1080, 1366x768
- Tablet: 768x1024 (iPad)
- Mobile: Not primary focus (basic responsive)

NFR19.3: Accessibility

- WCAG 2.1 Level AA compliance
- Keyboard navigation
- Screen reader support

- Color contrast ratios

NFR19.4: Security

- Role-based access control (RBAC)
- CSRF protection
- XSS prevention
- Secure session management

NFR19.5: Browser Support

- Chrome (latest 2 versions)
- Firefox (latest 2 versions)
- Safari (latest 2 versions)
- Edge (latest 2 versions)

Technical Architecture

Project Structure:

```
frontend/super-admin-portal/
├── public/
│   ├── favicon.ico
│   ├── logo.svg
│   └── images/
├── src/
│   ├── app/                    # Next.js 14 App Router
│   │   ├── layout.tsx          # Root layout
│   │   ├── page.tsx            # Dashboard (/)
│   │   ├── login/
│   │   │   └── page.tsx
│   │   ├── colleges/
│   │   │   ├── page.tsx        # List colleges
│   │   │   ├── new/
│   │   │   │   └── page.tsx    # Create college
│   │   │   └── [id]/
│   │   │       ├── page.tsx    # College details
```

```
│   │   │       └── edit/
│   │   │           └── page.tsx      # Edit college
│   │   ├── users/
│   │   │   ├── page.tsx            # List users
│   │   │   └── [id]/
│   │   │       └── page.tsx        # User details
│   │   ├── analytics/
│   │   │   ├── overview/
│   │   │   │   └── page.tsx
│   │   │   ├── colleges/
│   │   │   │   └── page.tsx
│   │   │   └── users/
│   │   │       └── page.tsx
│   │   ├── settings/
│   │   │   ├── general/
│   │   │   │   └── page.tsx
│   │   │   ├── email/
│   │   │   │   └── page.tsx
│   │   │   ├── security/
│   │   │   │   └── page.tsx
│   │   │   └── integrations/
│   │   │       └── page.tsx
│   │   └── audit-logs/
│   │       └── page.tsx
│   ├── components/
│   │   ├── ui/                     # Reusable UI components
│   │   │   ├── Button.tsx
│   │   │   ├── Input.tsx
│   │   │   ├── Table.tsx
│   │   │   ├── Card.tsx
│   │   │   ├── Modal.tsx
│   │   │   ├── Dropdown.tsx
│   │   │   ├── Tabs.tsx
│   │   │   └── Chart.tsx
│   │   ├── layout/
```

```
│   │   │   ├── Sidebar.tsx
│   │   │   ├── Header.tsx
│   │   │   ├── Footer.tsx
│   │   │   └── Breadcrumb.tsx
│   │   ├── dashboard/
│   │   │   ├── MetricCard.tsx
│   │   │   ├── ActivityFeed.tsx
│   │   │   └── QuickActions.tsx
│   │   ├── colleges/
│   │   │   ├── CollegeTable.tsx
│   │   │   ├── CollegeForm.tsx
│   │   │   └── CollegeCard.tsx
│   │   ├── users/
│   │   │   ├── UserTable.tsx
│   │   │   ├── UserFilters.tsx
│   │   │   └── UserActivityLog.tsx
│   │   └── analytics/
│   │       ├── LineChart.tsx
│   │       ├── BarChart.tsx
│   │       └── PieChart.tsx
│   ├── lib/
│   │   ├── api/                # API client functions
│   │   │   ├── colleges.ts
│   │   │   ├── users.ts
│   │   │   ├── analytics.ts
│   │   │   └── settings.ts
│   │   ├── auth.ts             # Authentication utilities
│   │   ├── utils.ts            # Helper functions
│   │   └── constants.ts        # Constants
│   ├── hooks/
│   │   ├── useAuth.ts
│   │   ├── useColleges.ts
│   │   ├── useUsers.ts
│   │   └── useAnalytics.ts
│   ├── store/                  # State management (Zustand)
```

```
│   │   ├── authStore.ts
│   │   ├── collegeStore.ts
│   │   └── uiStore.ts
│   ├── types/
│   │   ├── college.ts
│   │   ├── user.ts
│   │   ├── analytics.ts
│   │   └── api.ts
│   ├── styles/
│   │   └── globals.css              # Global styles (Tailwind)
│   └── middleware.ts                 # Auth middleware
├── .env.local
├── .eslintrc.json
├── next.config.js
├── package.json
├── tailwind.config.js
└── tsconfig.json
```

Tech Stack:

- **Framework**: Next.js 14 (App Router)
- **Language**: TypeScript
- **Styling**: Tailwind CSS
- **UI Components**: shadcn/ui (or custom)
- **State Management**: Zustand (or Redux Toolkit)
- **Forms**: React Hook Form + Zod validation
- **Charts**: Recharts or Chart.js
- **HTTP Client**: Axios or Fetch API
- **Authentication**: JWT tokens (stored in httpOnly cookies)
- **Icons**: Lucide React or Heroicons

API Integration:

```
// lib/api/colleges.ts
import axios from 'axios';
```

```
const API_BASE_URL = process.env.NEXT_PUBLIC_API_URL;

export const collegesApi = {
  getAll: async (params?: { page?: number; limit?: number; search?: string }) =
          > {
    const response = await axios.get(`${API_BASE_URL}/api/v1/colleges`,
          { params });
    return response.data;
  },

  getById: async (id: string) => {
    const response = await axios.get(`${API_BASE_URL}/api/v1/colleges/${id}`);
    return response.data;
  },

  create: async (data: CreateCollegeDto) => {
    const response = await axios.post(`${API_BASE_URL}/api/v1/colleges`,
          data);
    return response.data;
  },

  update: async (id: string, data: UpdateCollegeDto) => {
    const response = await axios.put(`${API_BASE_URL}/api/v1/colleges/${id}`,
          data);
    return response.data;
  },

  delete: async (id: string) => {
    await axios.delete(`${API_BASE_URL}/api/v1/colleges/${id}`);
  }
};
```

Authentication Flow:

```
// middleware.ts
import { NextResponse } from 'next/server';
```

```ts
import type { NextRequest } from 'next/server';

export function middleware(request: NextRequest) {
  const token = request.cookies.get('auth_token');

  if (!token && !request.nextUrl.pathname.startsWith('/login')) {
    return NextResponse.redirect(new URL('/login', request.url));
  }

  if (token && request.nextUrl.pathname.startsWith('/login')) {
    return NextResponse.redirect(new URL('/', request.url));
  }

  return NextResponse.next();
}

export const config = {
  matcher: ['/((?!api|_next/static|_next/image|favicon.ico).*)'],
};
```

Component Example:

```tsx
// components/colleges/CollegeTable.tsx
'use client';

import { useState } from 'react';
import { useColleges } from '@/hooks/useColleges';
import { Button } from '@/components/ui/Button';
import { Table } from '@/components/ui/Table';

export function CollegeTable() {
  const [page, setPage] = useState(1);
  const { colleges, isLoading, error } = useColleges({ page, limit: 20 });
```

```
  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error: {error.message}</div>;

  return (
    <div>
      <Table
        columns={[
          { key: 'name', label: 'College Name' },
          { key: 'code', label: 'Code' },
          { key: 'city', label: 'City' },
          { key: 'status', label: 'Status' },
          { key: 'actions', label: 'Actions' }
        ]}
        data={colleges.data}
        onRowClick={(college) => router.push(`/colleges/${college.id}`)}
      />
      <Pagination
        currentPage={page}
        totalPages={colleges.totalPages}
        onPageChange={setPage}
      />
    </div>
  );
}
```

Dependencies & Tech Stack

Core Dependencies:

```
{
  "dependencies": {
    "next": "^14.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
```

```
    "typescript": "^5.0.0",
    "tailwindcss": "^3.3.0",
    "axios": "^1.6.0",
    "zustand": "^4.4.0",
    "react-hook-form": "^7.48.0",
    "zod": "^3.22.0",
    "recharts": "^2.10.0",
    "lucide-react": "^0.294.0",
    "date-fns": "^2.30.0",
    "clsx": "^2.0.0",
    "tailwind-merge": "^2.0.0"
  },
  "devDependencies": {
    "@types/node": "^20.0.0",
    "@types/react": "^18.2.0",
    "eslint": "^8.54.0",
    "eslint-config-next": "^14.0.0",
    "prettier": "^3.1.0"
  }
}
```

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V19.1	Dashboard load	Browser	- Dashboard loads in <2s - All metrics displayed - Charts render correctly
V19.2	Create college	Manual	- Fill form, submit - College created in backend - Redirects to college details - Success notification shown
V19.3	User search	Manual	- Search by name - Results appear in <300ms

ID	Test Case	Tool	Acceptance Criteria
			- Filters work correctly - Pagination works
V19.4	Update settings	Manual	- Change platform name - Upload logo - Save settings - Changes reflected immediately
V19.5	Analytics charts	Browser	- Charts load with data - Date range filter works - Export to CSV works
V19.6	Responsive design	Browser DevTools	- Test on 1920x1080, 1366x768, 768x1024 - Layout adapts correctly - No horizontal scroll
V19.7	Authentication	Manual	- Login with valid credentials - Redirects to dashboard - Logout works - Protected routes redirect to login
V19.8	Accessibility	axe DevTools	- No critical accessibility issues - Keyboard navigation works - Screen reader compatible
V19.9	Form validation	Manual	- Submit empty form - Validation errors shown - Invalid email rejected - Required fields enforced
V19.10	Error handling	Manual	- Simulate API error - Error message displayed - Retry option available - No app crash

---

Phase 20: College Management Portal (Next.js)

Phase Goals

1. Build Next.js application for college admins, deans, and HODs
2. Implement college dashboard with department-level metrics
3. Create course and faculty management interfaces
4. Add student management with bulk operations
5. Build reports and analytics specific to college
6. Implement timetable and attendance management UI

User Stories

ID	User Story	Priority	Acceptance Criteria
US 20.1	As a **College Admin**, I want to see a dashboard with college metrics so that I can monitor operations	Must-Have	- Shows total students, faculty, courses - Attendance overview - Enrollment trends - Upcoming events/ deadlines
US 20.2	As an **HOD**, I want to manage courses in my department so that I can plan curriculum	Must-Have	- Create/edit courses - Assign teachers to subjects - Set prerequisites - Publish to catalog
US 20.3	As a **Dean**, I want to manage faculty so that I can track teaching assignments	Must-Have	- Add/edit faculty details - View teaching load - Assign to courses/ sections - Track performance

ID	User Story	Priority	Acceptance Criteria
US 20.4	As a **College Admin**, I want to bulk import students so that I can onboard new batches quickly	Must-Have	- Upload CSV file - Validate data - Preview before import - Track import progress - Download error report
US 20.5	As an **HOD**, I want to generate department reports so that I can analyze performance	Must-Have	- Select report type (attendance, grades, enrollment) - Apply filters (date, section, subject) - Preview and export - Schedule periodic reports
US 20.6	As a **Teacher**, I want to view and manage my timetable so that I know my schedule	Should-Have	- View weekly timetable - See room assignments - Mark attendance from timetable - Request room changes

Functional Requirements

FR20.1: College Dashboard

- Key metrics cards:
  - Total Students (with breakdown by year)
  - Total Faculty (teaching/non-teaching)
  - Active Courses
  - Average Attendance (last 7 days)
- Charts:
  - Enrollment trends (line chart, last 6 months)

    - Department-wise student distribution (pie chart)
    - Attendance trends (line chart, last 30 days)
- Widgets:
    - Upcoming events (exams, holidays, deadlines)
    - Recent activities
    - Announcements
    - Quick actions (Add Student, Create Course, Mark Attendance)

FR20.2: Department & Branch Management

- Pages:
    - `/departments` - List departments
    - `/departments/{id}` - Department details (courses, faculty, students)
    - `/branches` - List branches
    - `/branches/{id}` - Branch details
- Department details:
    - HOD information
    - Courses offered
    - Faculty list
    - Student count
    - Performance metrics

FR20.3: Course Management

- Pages:
    - `/courses` - List all courses (filterable by branch, year)
    - `/courses/new` - Create course
    - `/courses/{id}` - Course details
    - `/courses/{id}/edit` - Edit course
- Course form:
    - Basic info (name, code, credits, type)
    - Subjects (add multiple)
    - Prerequisites
    - Syllabus upload
    - Teacher assignment
- Course details:
    - Subject list with assigned teachers
    - Enrolled students
    - Attendance statistics

    - Grade distribution
    - Content library

FR20.4: Faculty Management

- Pages:
    - `/faculty` - List all faculty (table with filters)
    - `/faculty/new` - Add faculty
    - `/faculty/{id}` - Faculty profile
    - `/faculty/{id}/edit` - Edit faculty
- Faculty profile:
    - Personal details
    - Qualifications
    - Teaching assignments (subjects, sections)
    - Teaching load (hours/week)
    - Performance metrics (student feedback, attendance)
- Actions: Assign to course, View timetable, Send email

FR20.5: Student Management

- Pages:
    - `/students` - List all students (advanced filters)
    - `/students/new` - Add student
    - `/students/bulk-import` - Bulk import
    - `/students/{id}` - Student profile
    - `/students/{id}/edit` - Edit student
- Student filters: Batch, branch, section, status
- Student profile:
    - Personal details
    - Academic info (batch, section, enrollment date)
    - Enrolled courses
    - Attendance summary
    - Grade summary
    - Fee status
- Bulk import:
    - Upload CSV template
    - Validate data (show errors)
    - Preview import (first 10 rows)
    - Confirm and import

    - Track progress (progress bar)
    - Download error report

FR20.6: Timetable Management

- Pages:
    - `/timetable` - View timetable (weekly grid)
    - `/timetable/create` - Create timetable
    - `/timetable/rooms` - Room management
- Timetable views:
    - By section (student view)
    - By teacher (teacher view)
    - By room (room utilization)
- Features:
    - Drag-and-drop slots
    - Conflict detection (visual indicators)
    - Color-coded by subject
    - Export to PDF/iCal
    - Print view

FR20.7: Attendance Management

- Pages:
    - `/attendance` - Attendance dashboard
    - `/attendance/mark` - Mark attendance
    - `/attendance/reports` - Attendance reports
- Mark attendance:
    - Select class (course, section, date)
    - Student list with checkboxes
    - Bulk actions (mark all present/absent)
    - QR code generation
    - Save and notify students
- Attendance reports:
    - By student (individual attendance %)
    - By course (class-wise attendance)
    - Defaulter list (below threshold)
    - Export to CSV/PDF

FR20.8: Reports & Analytics

- Pages:
  - `/reports` - Reports dashboard
  - `/reports/attendance` - Attendance reports
  - `/reports/grades` - Grade reports
  - `/reports/enrollment` - Enrollment reports
  - `/reports/custom` - Custom report builder
- Report filters:
  - Date range
  - Department/branch
  - Course/section
  - Student/teacher
- Export formats: CSV, PDF, Excel
- Scheduled reports: Email delivery (daily/weekly/monthly)

FR20.9: Announcements & Notifications

- Pages:
  - `/announcements` - List announcements
  - `/announcements/new` - Create announcement
- Announcement form:
  - Title, content (rich text editor)
  - Target audience (all, specific branch/year)
  - Publish date/time
  - Attachments
- Notification channels: Email, push, in-app

FR20.10: Settings

- Pages:
  - `/settings/profile` - User profile
  - `/settings/college` - College settings
  - `/settings/academic` - Academic year, semesters
  - `/settings/notifications` - Notification preferences
- College settings:
  - College logo, name, address
  - Academic calendar
  - Attendance threshold

◦ Grading scheme

Non-Functional Requirements

NFR20.1: Performance

- Dashboard load: <2 seconds
- Student list (1000 students): <1 second
- Bulk import (5000 students): <5 minutes
- Report generation: <10 seconds

NFR20.2: Usability

- Intuitive navigation
- Consistent UI/UX across pages
- Helpful tooltips and hints
- Clear error messages

NFR20.3: Responsiveness

- Desktop: 1920x1080, 1366x768
- Tablet: 768x1024
- Mobile: 375x667 (basic support)

NFR20.4: Security

- Role-based access (College Admin, Dean, HOD, Teacher)
- Data encryption in transit
- Audit logging for critical actions

Technical Architecture

Project Structure:

```
frontend/college-portal/
├── public/
├── src/
│   ├── app/
│   │   ├── layout.tsx
```

```
│   │   ├── page.tsx              # Dashboard
│   │   ├── login/
│   │   ├── departments/
│   │   │   ├── page.tsx
│   │   │   └── [id]/
│   │   │       └── page.tsx
│   │   ├── branches/
│   │   ├── courses/
│   │   │   ├── page.tsx
│   │   │   ├── new/
│   │   │   │   └── page.tsx
│   │   │   └── [id]/
│   │   │       ├── page.tsx
│   │   │       └── edit/
│   │   │           └── page.tsx
│   │   ├── faculty/
│   │   │   ├── page.tsx
│   │   │   ├── new/
│   │   │   └── [id]/
│   │   ├── students/
│   │   │   ├── page.tsx
│   │   │   ├── new/
│   │   │   ├── bulk-import/
│   │   │   │   └── page.tsx
│   │   │   └── [id]/
│   │   ├── timetable/
│   │   │   ├── page.tsx
│   │   │   ├── create/
│   │   │   └── rooms/
│   │   ├── attendance/
│   │   │   ├── page.tsx
│   │   │   ├── mark/
│   │   │   └── reports/
│   │   ├── reports/
│   │   │   ├── page.tsx
```

```
│   │   │   ├── attendance/
│   │   │   ├── grades/
│   │   │   ├── enrollment/
│   │   │   └── custom/
│   │   ├── announcements/
│   │   │   ├── page.tsx
│   │   │   └── new/
│   │   └── settings/
│   │       ├── profile/
│   │       ├── college/
│   │       ├── academic/
│   │       └── notifications/
│   ├── components/
│   │   ├── ui/
│   │   ├── layout/
│   │   ├── dashboard/
│   │   │   ├── MetricCard.tsx
│   │   │   ├── EnrollmentChart.tsx
│   │   │   ├── AttendanceChart.tsx
│   │   │   └── UpcomingEvents.tsx
│   │   ├── courses/
│   │   │   ├── CourseTable.tsx
│   │   │   ├── CourseForm.tsx
│   │   │   └── SubjectList.tsx
│   │   ├── faculty/
│   │   │   ├── FacultyTable.tsx
│   │   │   ├── FacultyForm.tsx
│   │   │   └── TeachingLoad.tsx
│   │   ├── students/
│   │   │   ├── StudentTable.tsx
│   │   │   ├── StudentForm.tsx
│   │   │   ├── BulkImport.tsx
│   │   │   └── StudentProfile.tsx
│   │   ├── timetable/
│   │   │   ├── TimetableGrid.tsx
```

```
│   │   │   ├── TimeSlot.tsx
│   │   │   └── ConflictIndicator.tsx
│   │   ├── attendance/
│   │   │   ├── AttendanceSheet.tsx
│   │   │   ├── QRCodeGenerator.tsx
│   │   │   └── AttendanceReport.tsx
│   │   └── reports/
│   │       ├── ReportBuilder.tsx
│   │       ├── ReportPreview.tsx
│   │       └── ExportButton.tsx
│   ├── lib/
│   │   ├── api/
│   │   │   ├── courses.ts
│   │   │   ├── faculty.ts
│   │   │   ├── students.ts
│   │   │   ├── timetable.ts
│   │   │   ├── attendance.ts
│   │   │   └── reports.ts
│   │   └── utils/
│   │       ├── csv-parser.ts
│   │       ├── date-utils.ts
│   │       └── validators.ts
│   ├── hooks/
│   │   ├── useCourses.ts
│   │   ├── useFaculty.ts
│   │   ├── useStudents.ts
│   │   ├── useTimetable.ts
│   │   └── useAttendance.ts
│   ├── store/
│   │   ├── authStore.ts
│   │   ├── courseStore.ts
│   │   ├── studentStore.ts
│   │   └── uiStore.ts
│   └── types/
│       ├── course.ts
```

```
|       ├── faculty.ts
|       ├── student.ts
|       ├── timetable.ts
|       └── attendance.ts
├── package.json
├── next.config.js
└── tailwind.config.js
```

Key Components:

```tsx
// components/students/BulkImport.tsx
'use client';

import { useState } from 'react';
import { useStudents } from '@/hooks/useStudents';
import { Button } from '@/components/ui/Button';
import { FileUpload } from '@/components/ui/FileUpload';
import { ProgressBar } from '@/components/ui/ProgressBar';

export function BulkImport() {
  const [file, setFile] = useState<File | null>(null);
  const [validationErrors, setValidationErrors] = useState([]);
  const [importProgress, setImportProgress] = useState(0);
  const { bulkImport } = useStudents();

  const handleValidate = async () => {
    // Validate CSV file
    const errors = await validateCSV(file);
    setValidationErrors(errors);
  };

  const handleImport = async () => {
    const jobId = await bulkImport(file);
```

```
  // Poll for progress
  const interval = setInterval(async () => {
    const status = await getImportStatus(jobId);
    setImportProgress(status.progress);

    if (status.status === 'completed') {
      clearInterval(interval);
      // Show success message
    }
  }, 2000);
};

return (
  <div className="space-y-4">
    <FileUpload
      accept=".csv"
      onChange={setFile}
      label="Upload Student CSV"
    />

    {file && (
      <>
        <Button onClick={handleValidate}>Validate</Button>

        {validationErrors.length > 0 && (
          <ErrorList errors={validationErrors} />
        )}

        {validationErrors.length === 0 && (
          <Button onClick={handleImport}>Import Students</Button>
        )}
      </>
    )}

    {importProgress > 0 && (
```

```
        <ProgressBar value={importProgress} max={100} />
      )}
    </div>
  );
}
```

```
// components/timetable/TimetableGrid.tsx
'use client';

import { useTimetable } from '@/hooks/useTimetable';
import { TimeSlot } from './TimeSlot';

const DAYS = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'];
const TIME_SLOTS = ['09:00', '10:00', '11:00', '12:00', '14:00', '15:00', '16:00'];

export function TimetableGrid({ sectionId }: { sectionId: string }) {
  const { timetable, updateSlot } = useTimetable(sectionId);

  return (
    <div className="grid grid-cols-6 gap-2">
      <div className="font-bold">Time</div>
      {DAYS.map(day => (
        <div key={day} className="font-bold text-center">{day}</div>
      ))}

      {TIME_SLOTS.map(time => (
        <>
          <div key={time} className="font-semibold">{time}</div>
          {DAYS.map((day, dayIndex) => {
            const slot = timetable?.find(s => s.day === dayIndex + 1 && s.start
                Time === time);
            return (
              <TimeSlot
                key={`${day}-${time}`}
```

```
            slot={slot}
            onUpdate={(data) => updateSlot(slot?.id, data)}
          />
        );
      })}
    </>
  ))}
 </div>
 );
}
```

Dependencies & Tech Stack

Same as Phase 19, with additional:

- `react-dropzone`: File upload
- `papaparse`: CSV parsing
- `react-big-calendar`: Timetable calendar view (optional)
- `qrcode.react`: QR code generation

Verification Steps

ID	Test Case	Tool	Acceptance Criteria
V20.1	Dashboard load	Browser	- Loads in <2s, all metrics shown, charts render
V20.2	Create course	Manual	- Fill form, add subjects, assign teacher, save successfully
V20.3	Add faculty	Manual	- Fill form, upload photo, save, appears in list
V20.4	Bulk import students	Manual	- Upload CSV (1000 students), validate, import, track progress, success
			- Add slots, detect conflicts, save, export to PDF

ID	Test Case	Tool	Acceptance Criteria
V20.5	Create timetable	Manual	
V20.6	Mark attendance	Manual	- Select class, mark students, save, students notified
V20.7	Generate report	Manual	- Select attendance report, apply filters, preview, export to CSV
V20.8	Responsive design	DevTools	- Test on desktop, tablet, mobile, layout adapts
V20.9	Role-based access	Manual	- Login as HOD, can't access Dean-only features, proper error shown
V20.10	Search students	Manual	- Search by name, filter by batch, results in <300ms

---

End of Phases 19-20

Next Document: 06-PHASES-21-25.md for Student Portal, AWS Infrastructure, CI/CD, Mobile Apps, and AI/ML features.

LMS Platform PRD - Phases 21-25

Student Portal, Production Infrastructure & Advanced Features

---

Phase 21: Student LMS Portal (Next.js)

Phase Goals

1. Build Next.js application for students
2. Implement student dashboard with academic overview
3. Create course catalog and enrollment interface
4. Add assignment submission and grade viewing
5. Integrate real-time chat and notifications
6. Build mobile-responsive design

User Stories

ID	User Story	Priority	Acceptance Criteria
US 21.1	As a **Student**, I want to see my dashboard so that I can track my academic progress	Must-Have	- Shows enrolled courses, upcoming deadlines, recent grades - GPA and attendance % - Quick links to assignments, chat - Notifications feed
		Must-Have	- View course catalog with filters - See course details

ID	User Story	Priority	Acceptance Criteria
US 21.2	As a **Student**, I want to browse and enroll in courses so that I can register for the semester		(syllabus, teacher, schedule) - Enroll in courses - Join waitlist if full
US 21.3	As a **Student**, I want to submit assignments so that I can complete my coursework	Must-Have	- View assignment details and deadline - Upload files (PDF, DOCX, ZIP) - See submission confirmation - View grade and feedback after grading
US 21.4	As a **Student**, I want to chat with teachers and classmates so that I can collaborate	Must-Have	- Send/receive messages in real-time - Share files in chat - See typing indicators - Notifications for new messages
US 21.5	As a **Student**, I want to view my grades so that I can track my performance	Must-Have	- See all grades by course - View overall GPA - See grade trends (charts) - Download grade report
US 21.6	As a **Student**, I want to access course content so that I can study	Must-Have	- View lecture notes, videos, documents - Download materials - Organized by modules/ weeks - Track progress

Functional Requirements

FR21.1: Student Dashboard

- Widgets:
  - Academic summary (GPA, credits earned, attendance %)
  - Enrolled courses (cards with progress)
  - Upcoming deadlines (assignments, exams)
  - Recent grades
  - Notifications (last 10)
  - Quick actions (Submit Assignment, Join Class Chat)
- Charts:
  - GPA trend (line chart, semester-wise)
  - Attendance by course (bar chart)
- Calendar view: Upcoming classes, deadlines, exams

FR21.2: Course Catalog & Enrollment

- Pages:
  - `/courses` - Browse courses
  - `/courses/{id}` - Course details
  - `/my-courses` - Enrolled courses
- Course catalog:
  - Filters: Branch, year, type (core/elective), credits
  - Search by name, code, teacher
  - Sort by popularity, credits, name
- Course details:
  - Description, syllabus, learning outcomes
  - Teacher info
  - Schedule (days, time, room)
  - Prerequisites
  - Available seats
  - Enroll button
- Enrollment:
  - Check prerequisites
  - Confirm enrollment
  - Join waitlist if full
  - Email confirmation

FR21.3: My Courses

- Pages:
  - `/my-courses` - List of enrolled courses
  - `/my-courses/{id}` - Course home
  - `/my-courses/{id}/content` - Course materials
  - `/my-courses/{id}/assignments` - Assignments
  - `/my-courses/{id}/grades` - Grades
  - `/my-courses/{id}/chat` - Course chat
- Course home:
  - Announcements
  - Upcoming deadlines
  - Recent activity
  - Teacher contact
  - Syllabus download

FR21.4: Assignments

- Pages:
  - `/assignments` - All assignments (across courses)
  - `/assignments/{id}` - Assignment details
  - `/assignments/{id}/submit` - Submit assignment
- Assignment list:
  - Filter: Course, status (pending, submitted, graded)
  - Sort: Due date, course
  - Status badges (Pending, Submitted, Graded, Overdue)
- Assignment details:
  - Title, description, instructions
  - Due date (countdown timer)
  - Max marks
  - Attachments (reference files)
  - Submission status
- Submit assignment:
  - File upload (drag-and-drop)
  - Text submission (rich text editor)
  - Add comments
  - Submit button
  - Confirmation modal

FR21.5: Grades

- Pages:
  - `/grades` - All grades
  - `/grades/{courseId}` - Course-specific grades
- Grades view:
  - Table: Assignment, marks, max marks, percentage, date
  - Overall course grade
  - GPA calculation
  - Grade distribution chart
  - Download transcript (PDF)
- Grade details:
  - Marks breakdown (if rubric used)
  - Teacher feedback
  - View graded submission

FR21.6: Course Content

- Pages:
  - `/my-courses/{id}/content` - Content library
- Content organization:
  - Folders/modules (collapsible tree)
  - Files: PDFs, videos, documents
  - File preview (PDF viewer, video player)
  - Download button
  - Mark as completed (progress tracking)
- Video player:
  - Play/pause, seek, volume
  - Playback speed
  - Fullscreen
  - Subtitles (if available)

FR21.7: Chat

- Pages:
  - `/chat` - Chat interface
- Chat features:
  - Sidebar: List of conversations (course groups, direct messages)
  - Message area: Real-time messages

    - Input: Text input, file upload, emoji picker
    - Typing indicators
    - Read receipts
    - Search messages
    - Notifications

FR21.8: Attendance

- Pages:
    - `/attendance` - Attendance overview
    - `/attendance/{courseId}` - Course attendance
- Attendance view:
    - Overall attendance %
    - Course-wise breakdown
    - Calendar view (present/absent days)
    - Defaulter warning (if below threshold)
    - Regularization requests

FR21.9: Timetable

- Pages:
    - `/timetable` - Weekly timetable
- Timetable view:
    - Weekly grid (Monday-Friday)
    - Color-coded by course
    - Shows time, room, teacher
    - Export to iCal
    - Sync to Google Calendar

FR21.10: Profile & Settings

- Pages:
    - `/profile` - Student profile
    - `/settings` - Settings
- Profile:
    - Personal info (name, email, phone, photo)
    - Academic info (batch, branch, section, registration number)
    - Edit profile
- Settings:
    - Notification preferences (email, push, SMS)

- Password change
- Theme (light/dark mode)
- Language

Non-Functional Requirements

NFR21.1: Performance

- Dashboard load: <2 seconds
- Course catalog: <1 second
- Assignment submission: <3 seconds for 10MB file
- Chat message delivery: <100ms

NFR21.2: Mobile Responsiveness

- Mobile-first design
- Breakpoints: 375px, 768px, 1024px, 1440px
- Touch-friendly UI (large buttons, swipe gestures)
- Optimized for 3G/4G networks

NFR21.3: Accessibility

- WCAG 2.1 Level AA
- Keyboard navigation
- Screen reader support
- High contrast mode

NFR21.4: Offline Support

- Service worker for offline access
- Cache course content
- Offline message queue (send when online)
- Sync on reconnect

Technical Architecture

Project Structure:

```
frontend/student-portal/
├── public/
│   ├── service-worker.js
│   └── manifest.json
├── src/
│   ├── app/
│   │   ├── layout.tsx
│   │   ├── page.tsx                # Dashboard
│   │   ├── login/
│   │   ├── courses/
│   │   │   ├── page.tsx            # Course catalog
│   │   │   └── [id]/
│   │   │       └── page.tsx        # Course details
│   │   ├── my-courses/
│   │   │   ├── page.tsx
│   │   │   └── [id]/
│   │   │       ├── page.tsx        # Course home
│   │   │       ├── content/
│   │   │       ├── assignments/
│   │   │       ├── grades/
│   │   │       └── chat/
│   │   ├── assignments/
│   │   │   ├── page.tsx
│   │   │   └── [id]/
│   │   │       ├── page.tsx
│   │   │       └── submit/
│   │   ├── grades/
│   │   │   ├── page.tsx
│   │   │   └── [courseId]/
│   │   ├── chat/
│   │   │   └── page.tsx
│   │   ├── attendance/
│   │   │   ├── page.tsx
│   │   │   └── [courseId]/
```

```
│   │   ├── timetable/
│   │   │   └── page.tsx
│   │   ├── profile/
│   │   │   └── page.tsx
│   │   └── settings/
│   │       └── page.tsx
│   ├── components/
│   │   ├── ui/
│   │   ├── layout/
│   │   │   ├── Navbar.tsx
│   │   │   ├── Sidebar.tsx
│   │   │   ├── MobileNav.tsx
│   │   │   └── Footer.tsx
│   │   ├── dashboard/
│   │   │   ├── AcademicSummary.tsx
│   │   │   ├── EnrolledCourses.tsx
│   │   │   ├── UpcomingDeadlines.tsx
│   │   │   ├── RecentGrades.tsx
│   │   │   └── NotificationsFeed.tsx
│   │   ├── courses/
│   │   │   ├── CourseCard.tsx
│   │   │   ├── CourseFilters.tsx
│   │   │   ├── CourseDetails.tsx
│   │   │   └── EnrollButton.tsx
│   │   ├── assignments/
│   │   │   ├── AssignmentCard.tsx
│   │   │   ├── AssignmentDetails.tsx
│   │   │   ├── FileUpload.tsx
│   │   │   └── SubmissionForm.tsx
│   │   ├── grades/
│   │   │   ├── GradeTable.tsx
│   │   │   ├── GradeChart.tsx
│   │   │   └── GPACalculator.tsx
│   │   ├── content/
│   │   │   ├── ContentTree.tsx
```

```
│   │   │   ├── PDFViewer.tsx
│   │   │   ├── VideoPlayer.tsx
│   │   │   └── ProgressTracker.tsx
│   │   ├── chat/
│   │   │   ├── ChatSidebar.tsx
│   │   │   ├── MessageList.tsx
│   │   │   ├── MessageInput.tsx
│   │   │   └── TypingIndicator.tsx
│   │   └── attendance/
│   │       ├── AttendanceCalendar.tsx
│   │       ├── AttendanceChart.tsx
│   │       └── RegularizationForm.tsx
│   ├── lib/
│   │   ├── api/
│   │   │   ├── courses.ts
│   │   │   ├── assignments.ts
│   │   │   ├── grades.ts
│   │   │   ├── content.ts
│   │   │   ├── chat.ts
│   │   │   └── attendance.ts
│   │   ├── socket.ts          # Socket.IO client
│   │   └── sw-utils.ts        # Service worker utilities
│   ├── hooks/
│   │   ├── useCourses.ts
│   │   ├── useAssignments.ts
│   │   ├── useGrades.ts
│   │   ├── useChat.ts
│   │   └── useOffline.ts
│   ├── store/
│   │   ├── authStore.ts
│   │   ├── courseStore.ts
│   │   ├── chatStore.ts
│   │   └── offlineStore.ts
│   └── types/
```

```
├── package.json
└── next.config.js
```

Socket.IO Integration:

```typescript
// lib/socket.ts
import { io, Socket } from 'socket.io-client';

let socket: Socket | null = null;

export const initSocket = (token: string) => {
  socket = io(process.env.NEXT_PUBLIC_SOCKET_URL!, {
    auth: { token },
    transports: ['websocket']
  });

  socket.on('connect', () => {
    console.log('Connected to chat server');
  });

  socket.on('receive_message', (message) => {
    // Handle incoming message
    chatStore.addMessage(message);
  });

  return socket;
};

export const sendMessage = (roomId: string, content: string) => {
  socket?.emit('send_message', { roomId, content });
};
```

Service Worker (Offline Support):

```
// public/service-worker.js
const CACHE_NAME = 'lms-student-v1';
const urlsToCache = [
  '/',
  '/courses',
  '/my-courses',
  '/assignments',
  '/grades',
  '/chat',
  '/offline.html'
];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then((cache) => cache.addAll(urlsToCache))
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request)
      .then((response) => response || fetch(event.request))
  );
});
```

Dependencies & Tech Stack

Additional to Phase 19/20:

- `socket.io-client`: Real-time chat
- `react-pdf`: PDF viewer
- `video.js` or `plyr`: Video player
- `workbox`: Service worker utilities

- `react-calendar`: Calendar component
- `recharts`: Charts for grades/attendance

Verification Steps

ID	Test	Acceptance
V21.1	Dashboard load	Loads in <2s, all widgets displayed
V21.2	Enroll in course	Browse, select, enroll, confirmation shown
V21.3	Submit assignment	Upload file, submit, confirmation received
V21.4	Real-time chat	Send message, receive instantly, typing indicator works
V21.5	View grades	All grades shown, GPA calculated correctly
V21.6	Access content	View PDF, play video, download file
V21.7	Mobile responsive	Test on iPhone, Android, layout adapts
V21.8	Offline mode	Go offline, view cached content, send message when online
V21.9	Accessibility	Keyboard navigation, screen reader compatible
V21.10	Performance	Lighthouse score >90 for performance, accessibility

---

Phase 22: AWS Infrastructure (Terraform)

Phase Goals

1. Design production-ready AWS architecture
2. Create Terraform modules for all infrastructure
3. Set up EKS cluster for microservices

4. Configure RDS, MSK, S3, Elasticache
5. Implement monitoring with CloudWatch
6. Set up multi-environment (dev, staging, production)

Infrastructure Components

IC22.1: Networking (VPC)

- VPC with public and private subnets across 3 AZs
- NAT Gateways for private subnet internet access
- Internet Gateway for public subnets
- Route tables, security groups, NACLs
- VPC endpoints for S3, ECR (cost optimization)

IC22.2: Compute (EKS)

- EKS cluster (Kubernetes 1.28+)
- Node groups: On-demand and Spot instances
- Auto-scaling (Cluster Autoscaler or Karpenter)
- IAM roles for service accounts (IRSA)
- Pod security policies

IC22.3: Database (RDS)

- RDS PostgreSQL (Multi-AZ for production)
- Read replicas for read-heavy workloads
- Automated backups (7-day retention)
- Parameter groups for performance tuning
- Secrets Manager for credentials

IC22.4: Messaging (MSK)

- Amazon MSK (Managed Kafka)
- 3 brokers across 3 AZs
- Encryption in transit and at rest
- CloudWatch monitoring
- Auto-scaling storage

IC22.5: Caching (Elasticache)

- Elasticache for Redis (cluster mode)

- Multi-AZ with automatic failover
- Parameter groups for optimization
- CloudWatch alarms

IC22.6: Storage (S3)

- S3 buckets:
  - `lms-content-{env}`: Course content
  - `lms-assignments-{env}`: Assignments
  - `lms-backups-{env}`: Database backups
  - `lms-logs-{env}`: Application logs
- Lifecycle policies (transition to Glacier)
- Versioning enabled
- Server-side encryption (SSE-S3 or KMS)
- CloudFront CDN for content delivery

IC22.7: Monitoring & Logging

- CloudWatch Logs for application logs
- CloudWatch Metrics for infrastructure metrics
- CloudWatch Alarms for critical issues
- X-Ray for distributed tracing
- SNS for alert notifications

IC22.8: Security

- AWS WAF on ALB
- AWS Shield for DDoS protection
- Secrets Manager for secrets
- KMS for encryption keys
- IAM roles with least privilege
- Security groups (restrictive rules)

IC22.9: CI/CD Integration

- ECR for Docker images
- CodeBuild for image builds (optional)
- S3 for Terraform state (with locking via DynamoDB)

Terraform Structure

```
infrastructure/
├── terraform/
│   ├── modules/
│   │   ├── vpc/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   ├── outputs.tf
│   │   │   └── README.md
│   │   ├── eks/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   ├── outputs.tf
│   │   │   ├── node-groups.tf
│   │   │   └── iam.tf
│   │   ├── rds/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   ├── outputs.tf
│   │   │   └── parameter-group.tf
│   │   ├── msk/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   └── outputs.tf
│   │   ├── elasticache/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   └── outputs.tf
│   │   ├── s3/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   ├── outputs.tf
│   │   │   └── lifecycle.tf
```

```
│   │   ├── cloudwatch/
│   │   │   ├── main.tf
│   │   │   ├── alarms.tf
│   │   │   └── dashboards.tf
│   │   └── waf/
│   │       ├── main.tf
│   │       └── rules.tf
│   ├── environments/
│   │   ├── dev/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   ├── terraform.tfvars
│   │   │   └── backend.tf
│   │   ├── staging/
│   │   │   ├── main.tf
│   │   │   ├── variables.tf
│   │   │   ├── terraform.tfvars
│   │   │   └── backend.tf
│   │   └── production/
│   │       ├── main.tf
│   │       ├── variables.tf
│   │       ├── terraform.tfvars
│   │       └── backend.tf
│   └── README.md
└── kubernetes/
    ├── base/
    │   ├── namespace.yaml
    │   ├── configmap.yaml
    │   └── secrets.yaml
    ├── deployments/
    │   ├── auth-service.yaml
    │   ├── course-service.yaml
    │   ├── content-service.yaml
    │   └── ...
    ├── services/
```

```
    │   ├── auth-service-svc.yaml
    │   └── ...
    ├── ingress/
    │   ├── ingress.yaml
    │   └── alb-ingress-controller.yaml
    └── monitoring/
        ├── prometheus.yaml
        └── grafana.yaml
```

Example Terraform Module (EKS):

```
# modules/eks/main.tf
resource "aws_eks_cluster" "main" {
  name     = var.cluster_name
  role_arn = aws_iam_role.cluster.arn
  version  = var.kubernetes_version

  vpc_config {
    subnet_ids              = var.subnet_ids
    endpoint_private_access = true
    endpoint_public_access  = var.enable_public_access
    security_group_ids      = [aws_security_group.cluster.id]
  }

  enabled_cluster_log_types = ["api", "audit", "authenticator"]

  tags = var.tags
}

resource "aws_eks_node_group" "main" {
  cluster_name    = aws_eks_cluster.main.name
  node_group_name = "${var.cluster_name}-node-group"
  node_role_arn   = aws_iam_role.node.arn
  subnet_ids      = var.private_subnet_ids
```

```
  scaling_config {
    desired_size = var.desired_nodes
    max_size     = var.max_nodes
    min_size     = var.min_nodes
  }

  instance_types = var.instance_types
  capacity_type  = "ON_DEMAND"

  tags = var.tags
}
```

Kubernetes Deployment Example:

```yaml
# kubernetes/deployments/auth-service.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-service
  namespace: lms
spec:
  replicas: 3
  selector:
    matchLabels:
      app: auth-service
  template:
    metadata:
      labels:
        app: auth-service
    spec:
      containers:
      - name: auth-service
        image: <ECR_REPO>/auth-service:latest
```

```
    ports:
    - containerPort: 3000
    env:
    - name: DATABASE_URL
      valueFrom:
        secretKeyRef:
          name: db-credentials
          key: url
    - name: REDIS_URL
      valueFrom:
        configMapKeyRef:
          name: redis-config
          key: url
    resources:
      requests:
        memory: "256Mi"
        cpu: "250m"
      limits:
        memory: "512Mi"
        cpu: "500m"
    livenessProbe:
      httpGet:
        path: /health
        port: 3000
      initialDelaySeconds: 30
      periodSeconds: 10
    readinessProbe:
      httpGet:
        path: /health/ready
        port: 3000
      initialDelaySeconds: 10
      periodSeconds: 5
```

Cost Estimation (Monthly, Production)

Service	Configuration	Estimated Cost
EKS Cluster	1 cluster	$73
EC2 (EKS Nodes)	3x t3.large (on-demand)	$190
RDS PostgreSQL	db.t3.large (Multi-AZ)	$280
MSK	3 brokers (kafka.t3.small)	$240
Elasticache Redis	cache.t3.medium (2 nodes)	$100
S3	500 GB storage, 1 TB transfer	$50
CloudFront	1 TB data transfer	$85
ALB	1 load balancer	$25
NAT Gateway	2 gateways	$90
CloudWatch	Logs, metrics, alarms	$50
Total		**~$1,183/month**

Note: Costs can be optimized with Reserved Instances, Spot instances, and Savings Plans

Verification Steps

ID	Test	Acceptance
V22.1	Terraform plan	Plan succeeds, no errors
V22.2	Terraform apply	Infrastructure created successfully
V22.3	EKS cluster	Cluster accessible, nodes healthy
V22.4	RDS connectivity	Connect from EKS pod, query succeeds

ID	Test	Acceptance
V22.5	MSK connectivity	Produce/consume messages from EKS
V22.6	S3 access	Upload/download files from EKS
V22.7	Deploy services	All microservices deployed, healthy
V22.8	Load balancer	ALB routes traffic correctly
V22.9	Monitoring	CloudWatch logs/metrics visible
V22.10	Disaster recovery	Restore from backup, data intact

Phase 23: CI/CD Pipeline (GitHub Actions)

Phase Goals

1. Automate build, test, and deployment workflows
2. Implement multi-environment deployment (dev, staging, prod)
3. Add automated testing (unit, integration, E2E)
4. Set up Docker image builds and ECR pushes
5. Implement rollback mechanism
6. Add Slack/email notifications

CI/CD Workflows

Workflow 1: Build & Test

- Trigger: Push to any branch, pull request
- Steps:
    1. Checkout code
    2. Set up Node.js
    3. Install dependencies
    4. Run linter (ESLint)
    5. Run unit tests (Jest)
    6. Run integration tests

7. Upload test coverage
8. Build application
9. Notify on failure

Workflow 2: Build Docker Image

- Trigger: Push to `main`, `develop`, or release tags
- Steps:
    1. Checkout code
    2. Set up Docker Buildx
    3. Login to AWS ECR
    4. Build Docker image
    5. Tag image (commit SHA, branch name, `latest`)
    6. Push to ECR
    7. Scan image for vulnerabilities (Trivy)
    8. Notify on completion

Workflow 3: Deploy to Dev

- Trigger: Push to `develop` branch
- Steps:
    1. Checkout code
    2. Configure AWS credentials
    3. Update kubeconfig for EKS
    4. Update Kubernetes manifests (image tag)
    5. Apply manifests (`kubectl apply`)
    6. Wait for rollout completion
    7. Run smoke tests
    8. Notify on Slack

Workflow 4: Deploy to Staging

- Trigger: Push to `staging` branch or manual trigger
- Steps: Same as Dev, but with staging environment

Workflow 5: Deploy to Production

- Trigger: Manual approval after staging deployment
- Steps:
    1. Require manual approval (GitHub Environments)

2. Checkout code
3. Configure AWS credentials (production)
4. Update kubeconfig
5. Deploy with rolling update strategy
6. Monitor deployment (check pod health)
7. Run E2E tests
8. Rollback if tests fail
9. Notify on Slack/email

Workflow 6: Rollback

- Trigger: Manual trigger with deployment ID
- Steps:
  1. Get previous deployment revision
  2. Rollback Kubernetes deployment
  3. Verify rollback success
  4. Notify

Workflow 7: Database Migrations

- Trigger: Manual trigger or on deploy
- Steps:
  1. Checkout code
  2. Connect to RDS (via bastion or VPN)
  3. Run Prisma migrations
  4. Verify migration success
  5. Notify

GitHub Actions Structure

```
.github/
├── workflows/
│   ├── build-and-test.yml
│   ├── build-docker-image.yml
│   ├── deploy-dev.yml
│   ├── deploy-staging.yml
│   ├── deploy-production.yml
│   ├── rollback.yml
```

```
│   ├── database-migration.yml
│   └── security-scan.yml
├── actions/
│   ├── setup-node/
│   │   └── action.yml
│   ├── docker-build/
│   │   └── action.yml
│   └── deploy-k8s/
│       └── action.yml
└── CODEOWNERS
```

Example Workflow:

```yaml
# .github/workflows/deploy-production.yml
name: Deploy to Production

on:
  workflow_dispatch:
    inputs:
      service:
        description: 'Service to deploy'
        required: true
        type: choice
        options:
          - auth-service
          - course-service
          - all
      image_tag:
        description: 'Docker image tag'
        required: true

jobs:
  deploy:
    runs-on: ubuntu-latest
```

```yaml
    environment: production
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-1

      - name: Update kubeconfig
        run: |
          aws eks update-kubeconfig --name lms-production --region us-east-1

      - name: Deploy to Kubernetes
        run: |
          kubectl set image deployment/${{ inputs.service }} \
            ${{ inputs.service }}=${{ secrets.ECR_REGISTRY }}/${{ inputs.service }}:${
                {{ inputs.image_tag }} \
            -n lms
          kubectl rollout status deployment/${{ inputs.service }} -n lms --
                timeout=5m

      - name: Run smoke tests
        run: |
          ./scripts/smoke-tests.sh production ${{ inputs.service }}

      - name: Notify Slack
        if: always()
        uses: slackapi/slack-github-action@v1
        with:
          payload: |
            {
```

```
          "text": "Production deployment: ${{ job.status }}",
          "blocks": [
            {
              "type": "section",
              "text": {
                "type": "mrkdwn",
                "text": "*Service*: ${{ inputs.service }}\n*Status*: ${{ job.status }}
              \n*Tag*: ${{ inputs.image_tag }}"
              }
            }
          ]
        }
    env:
      SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
```

Verification Steps

ID	Test	Acceptance
V23.1	Build workflow	Triggers on push, tests pass, build succeeds
V23.2	Docker build	Image built, pushed to ECR, tagged correctly
V23.3	Deploy to dev	Deploys automatically on push to develop
V23.4	Deploy to staging	Manual trigger works, deployment succeeds
V23.5	Deploy to production	Requires approval, deploys successfully
V23.6	Rollback	Rollback to previous version works
V23.7	Notifications	Slack message sent on success/failure
V23.8	Security scan	Vulnerabilities detected and reported
V23.9	E2E tests	Tests run post-deployment, pass
V23.10	Database migration	Migrations run successfully

Phase 24: Mobile Applications (React Native)

Phase Goals

1. Build cross-platform mobile app (iOS & Android)
2. Implement core student features (courses, assignments, chat)
3. Add push notifications
4. Implement offline mode
5. Optimize for mobile performance
6. Publish to App Store and Play Store

Mobile App Features

F24.1: Authentication

- Login with email/password
- Biometric authentication (Face ID, Touch ID, fingerprint)
- Remember me (secure token storage)
- Logout

F24.2: Dashboard

- Academic summary (GPA, attendance)
- Enrolled courses (cards)
- Upcoming deadlines
- Recent grades
- Notifications

F24.3: Courses

- Browse courses
- Course details
- Enroll in courses
- View course content (PDFs, videos)
- Download for offline access

F24.4: Assignments

- List assignments (filter by course, status)

- Assignment details
- Submit assignment (file upload from gallery/camera)
- View submission status
- View grades and feedback

F24.5: Chat

- Real-time messaging
- Course group chats
- Direct messages with teachers
- File sharing (photos, documents)
- Push notifications for new messages

F24.6: Attendance

- View attendance summary
- QR code scanner for attendance
- Attendance calendar
- Regularization requests

F24.7: Grades

- View all grades
- Course-wise grades
- GPA calculation
- Grade trends (charts)

F24.8: Timetable

- Weekly timetable view
- Daily view
- Notifications for upcoming classes
- Add to device calendar

F24.9: Notifications

- Push notifications (FCM)
- In-app notifications
- Notification preferences
- Badge counts

F24.10: Profile & Settings

- View/edit profile
- Change password
- Notification settings
- Theme (light/dark)
- Logout

Technical Architecture

Project Structure:

```
mobile/
├── android/
│   ├── app/
│   └── build.gradle
├── ios/
│   ├── LMSApp/
│   └── Podfile
├── src/
│   ├── App.tsx
│   ├── navigation/
│   │   ├── AppNavigator.tsx
│   │   ├── AuthNavigator.tsx
│   │   └── TabNavigator.tsx
│   ├── screens/
│   │   ├── auth/
│   │   │   ├── LoginScreen.tsx
│   │   │   └── BiometricAuthScreen.tsx
│   │   ├── dashboard/
│   │   │   └── DashboardScreen.tsx
│   │   ├── courses/
│   │   │   ├── CoursesScreen.tsx
│   │   │   ├── CourseDetailsScreen.tsx
│   │   │   └── CourseContentScreen.tsx
```

```
│   │   ├── assignments/
│   │   │   ├── AssignmentsScreen.tsx
│   │   │   ├── AssignmentDetailsScreen.tsx
│   │   │   └── SubmitAssignmentScreen.tsx
│   │   ├── chat/
│   │   │   ├── ChatListScreen.tsx
│   │   │   └── ChatScreen.tsx
│   │   ├── attendance/
│   │   │   ├── AttendanceScreen.tsx
│   │   │   └── QRScannerScreen.tsx
│   │   ├── grades/
│   │   │   └── GradesScreen.tsx
│   │   ├── timetable/
│   │   │   └── TimetableScreen.tsx
│   │   └── profile/
│   │       ├── ProfileScreen.tsx
│   │       └── SettingsScreen.tsx
│   ├── components/
│   │   ├── common/
│   │   │   ├── Button.tsx
│   │   │   ├── Input.tsx
│   │   │   ├── Card.tsx
│   │   │   └── Loading.tsx
│   │   ├── courses/
│   │   │   └── CourseCard.tsx
│   │   ├── assignments/
│   │   │   └── AssignmentCard.tsx
│   │   └── chat/
│   │       ├── MessageBubble.tsx
│   │       └── ChatInput.tsx
│   ├── services/
│   │   ├── api/
│   │   │   ├── auth.ts
│   │   │   ├── courses.ts
│   │   │   ├── assignments.ts
```

```
│   │   │   ├── chat.ts
│   │   │   └── grades.ts
│   │   ├── socket.ts
│   │   ├── storage.ts
│   │   ├── notifications.ts
│   │   └── biometrics.ts
│   ├── store/
│   │   ├── authSlice.ts
│   │   ├── coursesSlice.ts
│   │   ├── chatSlice.ts
│   │   └── store.ts
│   ├── utils/
│   │   ├── constants.ts
│   │   ├── helpers.ts
│   │   └── validators.ts
│   └── types/
│       ├── auth.ts
│       ├── course.ts
│       └── assignment.ts
├── package.json
├── tsconfig.json
├── metro.config.js
└── app.json
```

Tech Stack:

- **Framework**: React Native 0.73+
- **Language**: TypeScript
- **Navigation**: React Navigation 6
- **State Management**: Redux Toolkit
- **HTTP Client**: Axios
- **Real-time**: Socket.IO client
- **Storage**: AsyncStorage, react-native-mmkv
- **Push Notifications**: @react-native-firebase/messaging
- **Biometrics**: react-native-biometrics
- **QR Scanner**: react-native-camera or react-native-vision-camera

- **PDF Viewer**: react-native-pdf
- **Video Player**: react-native-video
- **Charts**: react-native-chart-kit
- **UI Library**: React Native Paper or NativeBase

Key Implementation Examples:

```
// services/socket.ts
import io from 'socket.io-client';
import { store } from '../store/store';

export const socket = io(API_URL, {
  auth: {
    token: store.getState().auth.token
  },
  transports: ['websocket']
});

socket.on('receive_message', (message) => {
  store.dispatch(addMessage(message));
  // Show push notification if app in background
});
```

```
// screens/assignments/SubmitAssignmentScreen.tsx
import { launchImageLibrary } from 'react-native-image-picker';

const SubmitAssignmentScreen = ({ route }) => {
  const [file, setFile] = useState(null);

  const pickFile = async () => {
    const result = await launchImageLibrary({
      mediaType: 'mixed',
      selectionLimit: 1
    });
```

```
    if (result.assets) {
      setFile(result.assets[0]);
    }
  };

  const submitAssignment = async () => {
    const formData = new FormData();
    formData.append('file', {
      uri: file.uri,
      type: file.type,
      name: file.fileName
    });

    await api.submitAssignment(assignmentId, formData);
    // Show success message
  };

  return (
    <View>
      <Button onPress={pickFile}>Choose File</Button>
      {file && <Text>{file.fileName}</Text>}
      <Button onPress={submitAssignment}>Submit</Button>
    </View>
  );
};
```

App Store Deployment

iOS (App Store):

1. Create Apple Developer account ($99/year)
2. Configure app in App Store Connect
3. Generate certificates and provisioning profiles
4. Build release version (`npx react-native run-ios --configuration Release`)
5. Archive and upload to App Store Connect

6. Submit for review
7. Approval (1-3 days)

Android (Play Store):

1. Create Google Play Developer account ($25 one-time)
2. Generate signing key
3. Build release APK/AAB (`cd android && ./gradlew bundleRelease`)
4. Upload to Play Console
5. Fill app details, screenshots
6. Submit for review
7. Approval (1-3 days)

Verification Steps

ID	Test	Acceptance
V24.1	Login	Login with credentials, biometric auth works
V24.2	Dashboard	Loads in <2s, shows correct data
V24.3	Enroll in course	Browse, enroll, confirmation shown
V24.4	Submit assignment	Pick file, upload, submission successful
V24.5	Real-time chat	Send/receive messages instantly
V24.6	Push notifications	Receive notification when app in background
V24.7	QR scanner	Scan QR code, attendance marked
V24.8	Offline mode	View cached content offline
V24.9	Performance	App runs smoothly on mid-range devices
V24.10	App store	App approved and published

Phase 25: AI/ML Features

Phase Goals

1. Implement personalized course recommendations
2. Add AI-powered essay grading
3. Build online exam proctoring system
4. Create AI chatbot for student queries
5. Implement predictive analytics for at-risk students

AI/ML Features

ML25.1: Course Recommendation Engine

- Algorithm: Collaborative filtering + content-based filtering
- Input: Student profile, past enrollments, grades, interests
- Output: Top 10 recommended courses with scores
- Model: Matrix factorization (ALS) or neural collaborative filtering
- Training: Weekly on enrollment and grade data
- Endpoint: `GET /api/v1/recommendations/courses`

ML25.2: AI-Powered Essay Grading

- Model: Fine-tuned BERT or GPT for essay scoring
- Features:
  - Grammar and spelling check
  - Coherence and structure analysis
  - Content relevance scoring
  - Plagiarism detection (similarity with corpus)
- Output: Score (0-100), feedback, highlighted issues
- Human-in-the-loop: Teacher reviews AI grades
- Endpoint: `POST /api/v1/ai/grade-essay`

ML25.3: Online Exam Proctoring

- Features:
  - Face detection (ensure student present)
  - Multiple face detection (flag if >1 person)

    - Gaze tracking (flag if looking away)
    - Tab switching detection (flag if leaves exam tab)
    - Audio monitoring (flag if speaking)
- Model: Face detection (MTCNN), object detection (YOLO)
- Real-time processing: WebRTC stream to backend
- Dashboard: Proctor views flagged events
- Endpoint: `POST /api/v1/proctoring/analyze-frame`

ML25.4: AI Chatbot

- Model: Fine-tuned GPT-3.5 or LLaMA on LMS FAQs
- Capabilities:
    - Answer course-related questions
    - Explain assignment instructions
    - Provide deadline reminders
    - Guide through platform features
- Integration: Chat interface, voice input (optional)
- Fallback: Escalate to human support if confidence <70%
- Endpoint: `POST /api/v1/chatbot/query`

ML25.5: Predictive Analytics (At-Risk Students)

- Model: Gradient boosting (XGBoost) or neural network
- Features:
    - Attendance %
    - Assignment submission rate
    - Grade trends
    - Login frequency
    - Time spent on platform
- Output: Risk score (0-100), risk category (low/medium/high)
- Intervention: Alert teachers/counselors for high-risk students
- Training: Monthly on historical data
- Endpoint: `GET /api/v1/analytics/at-risk-students`

Technical Architecture

```
services/ai-service/
├── Dockerfile
```

```
├── requirements.txt
├── src/
│   ├── app.py                    # Flask/FastAPI app
│   ├── config/
│   │   └── settings.py
│   ├── controllers/
│   │   ├── recommendation_controller.py
│   │   ├── grading_controller.py
│   │   ├── proctoring_controller.py
│   │   ├── chatbot_controller.py
│   │   └── analytics_controller.py
│   ├── services/
│   │   ├── recommendation_service.py
│   │   ├── essay_grading_service.py
│   │   ├── proctoring_service.py
│   │   ├── chatbot_service.py
│   │   └── risk_prediction_service.py
│   ├── models/
│   │   ├── recommendation_model.py
│   │   ├── essay_grading_model.py
│   │   ├── face_detection_model.py
│   │   └── risk_model.py
│   └── utils/
│       ├── data_preprocessing.py
│       └── model_loader.py
├── ml/
│   ├── notebooks/
│   │   ├── recommendation_training.ipynb
│   │   ├── essay_grading_training.ipynb
│   │   └── risk_prediction_training.ipynb
│   ├── datasets/
│   │   ├── enrollments.csv
│   │   ├── essays_graded.csv
│   │   └── student_performance.csv
│   └── trained_models/
```

```
|       ├── recommendation_model.pkl
|       ├── essay_grading_model.pt
|       └── risk_model.pkl
└── tests/
```

Tech Stack:

- **Framework**: FastAPI or Flask
- **ML Libraries**: scikit-learn, TensorFlow/PyTorch, Transformers (Hugging Face)
- **Face Detection**: OpenCV, MTCNN, dlib
- **NLP**: spaCy, NLTK, Transformers
- **Deployment**: Docker, GPU instances (for deep learning)

Example Implementation:

```python
# services/recommendation_service.py
import numpy as np
from sklearn.decomposition import NMF
import pickle

class RecommendationService:
    def __init__(self):
        with open('ml/trained_models/recommendation_model.pkl', 'rb') as f:
            self.model = pickle.load(f)
        self.course_features = self.load_course_features()

    def get_recommendations(self, student_id, top_n=10):
        # Get student's enrollment history
        enrollments = self.get_student_enrollments(student_id)

        # Create user-item matrix
        user_vector = self.create_user_vector(enrollments)

        # Predict scores for all courses
        scores = self.model.transform(user_vector)
```

```
        # Get top N courses
        top_indices = np.argsort(scores[0])[-top_n:][::-1]
        recommendations = [
            {
                'course_id': self.course_features[i]['id'],
                'name': self.course_features[i]['name'],
                'score': float(scores[0][i])
            }
            for i in top_indices
        ]

        return recommendations
```

```
# services/proctoring_service.py
import cv2
import numpy as np
from mtcnn import MTCNN

class ProctoringService:
    def __init__(self):
        self.face_detector = MTCNN()

    def analyze_frame(self, frame_data):
        # Decode base64 image
        img = self.decode_image(frame_data)

        # Detect faces
        faces = self.face_detector.detect_faces(img)

        flags = []

        # Check number of faces
        if len(faces) == 0:
            flags.append({'type': 'no_face', 'severity': 'high'})
```

```
        elif len(faces) > 1:
            flags.append({'type': 'multiple_faces', 'severity': 'high'})

        # Check gaze direction (simplified)
        if len(faces) == 1:
            face = faces[0]
            # Analyze eye position, head pose
            if self.is_looking_away(face):
                flags.append({'type': 'looking_away', 'severity': 'medium'})

        return {
            'timestamp': time.time(),
            'faces_detected': len(faces),
            'flags': flags
        }
```

Model Training Pipeline

1. **Data Collection**: Gather training data from production
2. **Data Preprocessing**: Clean, normalize, feature engineering
3. **Model Training**: Train on historical data
4. **Evaluation**: Validate on test set, measure metrics (accuracy, RMSE, F1)
5. **Deployment**: Save model, deploy to AI service
6. **Monitoring**: Track model performance, retrain if drift detected

Verification Steps

ID	Test	Acceptance
V25.1	Course recommendations	Returns relevant courses, accuracy >70%
V25.2	Essay grading	AI grade within ±10% of human grade
V25.3	Proctoring	Detects faces, flags violations correctly
V25.4	Chatbot	Answers FAQs correctly, confidence >80%

ID	Test	Acceptance
V25.5	At-risk prediction	Identifies at-risk students, precision >75%
V25.6	Model latency	Recommendation <500ms, grading <5s
V25.7	Model retraining	Retraining pipeline runs successfully
V25.8	A/B testing	AI features improve engagement by >15%

End of Phases 21-25

Congratulations! You have completed the comprehensive PRD for all 25 phases of the LMS Platform. This document provides a complete roadmap from foundation to advanced AI/ML features, covering:

- **Backend Services**: 10+ microservices
- **Frontend Applications**: 3 web portals + mobile app
- **Infrastructure**: Production-ready AWS deployment
- **CI/CD**: Automated pipelines
- **Advanced Features**: AI/ML, real-time chat, analytics

Next Steps:

1. Review and refine PRD with stakeholders
2. Prioritize phases based on business value
3. Allocate resources and create sprint plans
4. Begin implementation starting with Phase 1
5. Iterate based on feedback and metrics

Total Estimated Timeline: 2-3 years for full implementation across all 25 phases.

LMS Platform PRD - Appendix

Reference Materials & Technical Specifications

A. Complete Data Models

This section provides a comprehensive reference of all database models across the LMS platform.

A.1 Authentication & User Management

```
// User Models (Polymorphic - different tables per role)

model Organization {
  id             String   @id @default(uuid())
  name           String
  email          String   @unique
  phone          String?
  address        String?
  website        String?
  logo           String?  // S3 URL
  isActive       Boolean  @default(true)
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt

  colleges       College[]
}
```

```
model College {
  id             String   @id @default(uuid())
  organizationId  String
  name           String
  code           String   @unique
  email          String   @unique
  phone          String
  address        String
  city           String
  state          String
  country        String
  pincode        String
  logo           String?
  timezone       String   @default("Asia/Kolkata")
  isActive       Boolean  @default(true)
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt

  organization   Organization @relation(fields: [organizationId], references:
[id])
  branches       Branch[]

  @@index([organizationId])
}

model Dean {
  id             String   @id @default(uuid())
  collegeId      String
  name           String
  email          String   @unique
  phone          String
  passwordHash   String
  photo          String?
  isActive       Boolean  @default(true)
  createdAt      DateTime @default(now())
```

```
  updatedAt       DateTime @updatedAt

  @@index([collegeId])
}

model HOD {
  id              String   @id @default(uuid())
  collegeId       String
  branchId        String
  name            String
  email           String   @unique
  phone           String
  passwordHash    String
  photo           String?
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  @@index([collegeId, branchId])
}

model Teacher {
  id              String   @id @default(uuid())
  collegeId       String
  branchId        String?
  name            String
  email           String   @unique
  phone           String
  passwordHash    String
  photo           String?
  qualifications  String?
  specialization  String?
  experience      Int?     // Years
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
```

```
  updatedAt       DateTime @updatedAt

  @@index([collegeId])
}

model Student {
  id              String   @id @default(uuid())
  collegeId       String
  branchId        String
  batchId         String
  sectionId       String?
  registrationNo  String   @unique
  name            String
  email           String   @unique
  phone           String
  passwordHash    String
  photo           String?
  dateOfBirth     DateTime?
  gender          String?
  address         String?
  guardianName    String?
  guardianPhone   String?
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  @@index([collegeId, branchId, batchId])
  @@index([registrationNo])
}

model NonTeachingStaff {
  id              String   @id @default(uuid())
  collegeId       String
  name            String
  email           String   @unique
```

```
  phone          String
  passwordHash   String
  role           String   // "librarian", "accountant", "clerk"
  photo          String?
  isActive       Boolean  @default(true)
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt

  @@index([collegeId])
}

// Session Management
model UserSession {
  id             String   @id @default(uuid())
  userId         String
  userType       String
  sessionToken   String   @unique
  deviceId       String
  deviceType     String?
  browser        String?
  os             String?
  ipAddress      String
  location       String?
  createdAt      DateTime @default(now())
  expiresAt      DateTime
  invalidatedAt  DateTime?
  isActive       Boolean  @default(true)

  @@index([userId, userType, isActive])
  @@index([sessionToken])
}

// Password Management
model PasswordHistory {
  id             String   @id @default(uuid())
```

```
  userId          String
  userType        String
  passwordHash    String
  createdAt       DateTime @default(now())

  @@index([userId, userType])
}

// API Keys
model ApiKey {
  id              String   @id @default(uuid())
  keyHash         String   @unique
  name            String
  description     String?
  createdBy       String
  createdByType   String
  scopes          String[]
  rateLimit       Int      @default(100)
  isActive        Boolean  @default(true)
  expiresAt       DateTime?
  lastUsedAt      DateTime?
  createdAt       DateTime @default(now())

  @@index([keyHash])
}
```

A.2 Academic Structure

```
model Branch {
  id              String   @id @default(uuid())
  collegeId       String
  name            String   // "Computer Science", "Mechanical"
  code            String   // "CSE", "MECH"
  description     String?
```

```
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  college         College  @relation(fields: [collegeId], references: [id])
  courses         Course[]
  batches         Batch[]

  @@unique([collegeId, code])
  @@index([collegeId])
}

model Batch {
  id              String   @id @default(uuid())
  branchId        String
  year            String   // "2024", "2025"
  startDate       DateTime
  endDate         DateTime
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())

  branch          Branch   @relation(fields: [branchId], references: [id])
  sections        Section[]

  @@unique([branchId, year])
  @@index([branchId])
}

model Section {
  id              String   @id @default(uuid())
  batchId         String
  name            String   // "A", "B", "C"
  capacity        Int      @default(60)
  currentStrength Int      @default(0)
  createdAt       DateTime @default(now())
```

```
  batch          Batch    @relation(fields: [batchId], references: [id])

  @@unique([batchId, name])
  @@index([batchId])
}
```

A.3 Course Management

```
model Course {
  id             String    @id @default(uuid())
  code           String    @unique
  name           String
  branchId       String
  credits        Int
  description    String?
  type           String    // "core", "elective"
  academicYear   String
  semester       Int?
  isActive       Boolean   @default(true)
  createdBy      String
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
  deletedAt      DateTime?

  subjects       Subject[]
  prerequisites  CoursePrerequisite[] @relation("CoursePrerequisites")
  prerequisiteFor CoursePrerequisite[] @relation("PrerequisiteFor")
  enrollments    Enrollment[]

  @@index([branchId, academicYear])
}

model Subject {
```

```
  id              String   @id @default(uuid())
  courseId        String
  code            String
  name            String
  credits         Int
  type            String   // "theory", "lab", "project"
  syllabus        String?  @db.Text
  learningOutcomes String? @db.Text
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  course          Course   @relation(fields: [courseId], references: [id])
  teacherAssignments TeacherSubjectAssignment[]

  @@unique([courseId, code])
  @@index([courseId])
}

model TeacherSubjectAssignment {
  id              String   @id @default(uuid())
  teacherId       String
  subjectId       String
  sectionId       String
  academicYear    String
  semester        Int
  assignedAt      DateTime @default(now())
  assignedBy      String

  subject         Subject  @relation(fields: [subjectId], references: [id])

  @@unique([teacherId, subjectId, sectionId, academicYear, semester])
  @@index([teacherId])
  @@index([subjectId])
}
```

```
model CoursePrerequisite {
  id              String   @id @default(uuid())
  courseId        String
  prerequisiteId  String
  isStrict        Boolean  @default(true)
  createdAt       DateTime @default(now())

  course          Course   @relation("CoursePrerequisites", fields: [courseId],
references: [id])
  prerequisite    Course   @relation("PrerequisiteFor", fields: [prerequisiteId],
references: [id])

  @@unique([courseId, prerequisiteId])
}

model Enrollment {
  id              String   @id @default(uuid())
  studentId       String
  courseId        String
  sectionId       String?
  academicYear    String
  semester        Int
  status          String   // "enrolled", "waitlisted", "dropped"
  enrolledAt      DateTime @default(now())
  allocatedAt     DateTime?
  droppedAt       DateTime?
  grade           String?

  course          Course   @relation(fields: [courseId], references: [id])

  @@unique([studentId, courseId, academicYear, semester])
  @@index([studentId])
  @@index([courseId])
}
```

```
model Waitlist {
  id             String   @id @default(uuid())
  studentId      String
  courseId       String
  position       Int
  joinedAt       DateTime @default(now())
  notifiedAt     DateTime?
  expiresAt      DateTime?

  @@unique([studentId, courseId])
  @@index([courseId, position])
}
```

A.4 Content & Assignments

```
model Content {
  id             String   @id @default(uuid())
  title          String
  description    String?
  type           String   // "document", "video", "image", "audio"
  mimeType       String
  size           Int
  duration       Int?
  s3Key          String   @unique
  s3Bucket       String
  courseId       String
  subjectId      String?
  folderId       String?
  uploadedBy     String
  uploadedByType String
  isPublished    Boolean  @default(false)
  publishedAt    DateTime?
  version        Int      @default(1)
```

```
  tags          String[]
  position      Int?
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
  deletedAt     DateTime?

  @@index([courseId])
  @@index([folderId])
}

model Folder {
  id            String   @id @default(uuid())
  name          String
  courseId      String
  parentId      String?
  position      Int
  createdBy     String
  createdAt     DateTime @default(now())

  @@index([courseId])
  @@index([parentId])
}

model Assignment {
  id            String   @id @default(uuid())
  title         String
  description   String   @db.Text
  courseId      String
  subjectId     String
  sectionId     String?
  createdBy     String
  dueDate       DateTime
  maxMarks      Int
  submissionType  String   // "file", "text", "code", "mcq"
  allowLate     Boolean  @default(false)
```

```
  lateDeadline    DateTime?
  latePenaltyPerDay Int?
  attachments     String[]
  rubric          Json?
  status          String   @default("draft")
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  submissions     Submission[]

  @@index([courseId, dueDate])
  @@index([createdBy])
}

model Submission {
  id              String   @id @default(uuid())
  assignmentId    String
  studentId       String
  files           String[]
  textContent     String?  @db.Text
  codeContent     String?  @db.Text
  submittedAt     DateTime @default(now())
  isLate          Boolean  @default(false)
  lateDays        Int?
  status          String   @default("submitted")
  plagiarismScore Float?
  flaggedForReview Boolean @default(false)

  assignment      Assignment @relation(fields: [assignmentId], references: [id])
  grade           Grade?

  @@unique([assignmentId, studentId])
  @@index([studentId])
}
```

```
model Grade {
  id             String   @id @default(uuid())
  submissionId   String   @unique
  marks          Float
  maxMarks       Int
  finalMarks     Float
  feedback       String?  @db.Text
  rubricScores   Json?
  gradedBy       String
  gradedAt       DateTime @default(now())
  status         String   @default("draft")
  releasedAt     DateTime?

  submission     Submission @relation(fields: [submissionId], references: [id])

  @@index([submissionId])
}
```

A.5 Attendance & Grading

```
model AttendanceSession {
  id             String   @id @default(uuid())
  courseId       String
  subjectId      String
  sectionId      String
  teacherId      String
  scheduledAt    DateTime
  duration       Int
  qrCode         String?  @unique
  qrExpiresAt    DateTime?
  location       Json?
  status         String   // "scheduled", "ongoing", "completed"
  createdAt      DateTime @default(now())
```

```
  attendances     Attendance[]

  @@index([courseId, scheduledAt])
  @@index([teacherId])
}

model Attendance {
  id              String   @id @default(uuid())
  sessionId       String
  studentId       String
  status          String   // "present", "absent", "late", "excused"
  markedAt        DateTime @default(now())
  markedBy        String?
  method          String   // "manual", "qr", "geofence"
  location        Json?

  session         AttendanceSession @relation(fields: [sessionId], references: [id])
  regularization  AttendanceRegularization?

  @@unique([sessionId, studentId])
  @@index([studentId])
}

model AttendanceRegularization {
  id              String   @id @default(uuid())
  attendanceId    String   @unique
  studentId       String
  reason          String   @db.Text
  documents       String[]
  status          String   @default("pending")
  submittedAt     DateTime @default(now())
  reviewedBy      String?
  reviewedAt      DateTime?
  comments        String?
```

```
  attendance      Attendance @relation(fields: [attendanceId], references: [id])

  @@index([studentId, status])
}

model MCQTest {
  id              String   @id @default(uuid())
  title           String
  courseId        String
  subjectId       String
  duration        Int
  totalMarks      Int
  questions       Json
  createdBy       String
  scheduledAt     DateTime?
  dueDate         DateTime?
  status          String   @default("draft")
  createdAt       DateTime @default(now())

  attempts        MCQAttempt[]

  @@index([courseId])
}

model MCQAttempt {
  id              String   @id @default(uuid())
  testId          String
  studentId       String
  answers         Json
  score           Float
  maxScore        Int
  startedAt       DateTime @default(now())
  submittedAt     DateTime?

  test            MCQTest  @relation(fields: [testId], references: [id])
```

```
  @@unique([testId, studentId])
  @@index([studentId])
}
```

A.6 Communication

```
// MongoDB Schema (Mongoose)
const roomSchema = new Schema({
  _id: String,
  name: String,
  type: { type: String, enum: ['direct', 'group', 'course'] },
  members: [{
    userId: String,
    userType: String,
    joinedAt: Date
  }],
  createdBy: String,
  createdAt: { type: Date, default: Date.now },
  lastMessageAt: Date,
  metadata: Schema.Types.Mixed
});

const messageSchema = new Schema({
  roomId: { type: String, index: true },
  senderId: { type: String, index: true },
  senderName: String,
  senderType: String,
  content: { type: String, text: true },
  type: { type: String, enum: ['text', 'file', 'image'] },
  fileUrl: String,
  fileName: String,
  fileSize: Number,
  timestamp: { type: Date, default: Date.now, index: true },
```

```
  readBy: [String],
  editedAt: Date,
  deletedAt: Date
});
```

A.7 Analytics & Reporting

```
model BulkImportJob {
  id              String   @id @default(uuid())
  type            String
  fileUrl         String
  fileName        String
  uploadedBy      String
  uploadedByType  String
  collegeId       String?
  status          String
  totalRows       Int?
  successRows     Int      @default(0)
  failedRows      Int      @default(0)
  errorReportUrl  String?
  options         Json?
  startedAt       DateTime?
  completedAt     DateTime?
  createdAt       DateTime @default(now())

  @@index([status, createdAt])
}

model NotificationLog {
  id              String   @id @default(uuid())
  userId          String
  type            String
  channel         String   // "email", "sms", "push"
  status          String   // "sent", "failed", "delivered"
```

```
  sentAt          DateTime @default(now())
  deliveredAt     DateTime?
  error           String?

  @@index([userId, sentAt])
}
```

A.8 System Configuration

```
model Timetable {
  id              String   @id @default(uuid())
  name            String
  academicYear    String
  semester        Int
  branchId        String
  createdBy       String
  status          String   @default("draft")
  createdAt       DateTime @default(now())

  slots           TimetableSlot[]
}

model TimetableSlot {
  id              String   @id @default(uuid())
  timetableId     String
  day             Int
  startTime       String
  endTime         String
  subjectId       String
  teacherId       String
  roomId          String
  sectionId       String
  type            String
  isRecurring     Boolean  @default(true)
```

```
  exceptions      Json?

  timetable       Timetable @relation(fields: [timetableId], references: [id])

  @@index([timetableId, day])
}

model Room {
  id              String   @id @default(uuid())
  name            String
  building        String
  capacity        Int
  type            String
  facilities      String[]
}

model Book {
  id              String   @id @default(uuid())
  isbn            String   @unique
  title           String
  author          String
  publisher       String?
  category        String
  totalCopies     Int
  availableCopies Int
  location        String
  createdAt       DateTime @default(now())

  transactions    BookTransaction[]
  reservations    BookReservation[]
}

model FeeStructure {
  id              String   @id @default(uuid())
  name            String
```

```
  courseId        String?
  batchId         String?
  components      Json
  totalAmount     Float
  dueDate         DateTime
  createdAt       DateTime @default(now())
}

model Payment {
  id              String   @id @default(uuid())
  studentId       String
  feeStructureId  String
  amount          Float
  gateway         String
  orderId         String   @unique
  transactionId   String?
  status          String
  paidAt          DateTime?
  receiptUrl      String?
  createdAt       DateTime @default(now())

  @@index([studentId])
}
```

B. API Contracts

B.1 Authentication APIs

POST /api/v1/auth/organization/register

Request:

```
{
  "name": "ABC Education Group",
```

```
  "email": "admin@abcedu.com",
  "password": "SecurePass123!",
  "phone": "+91-9876543210",
  "address": "123 Main St, City"
}
```

Response (201):

```
{
  "success": true,
  "data": {
    "id": "uuid",
    "name": "ABC Education Group",
    "email": "admin@abcedu.com"
  },
  "message": "Organization registered successfully. Please verify your email."
}
```

POST /api/v1/auth/student/login

Request:

```
{
  "email": "student@example.com",
  "password": "password123"
}
```

Response (200):

```
{
  "success": true,
  "data": {
    "user": {
      "id": "uuid",
      "name": "John Doe",
      "email": "student@example.com",
      "role": "student"
```

```
    },
    "accessToken": "paseto.token.here",
    "refreshToken": "refresh.token.here"
  }
}
```

B.2 Course APIs

GET /api/v1/courses?branch={branchId}&page=1&limit=20

Response (200):

```
{
  "success": true,
  "data": {
    "courses": [
      {
        "id": "uuid",
        "code": "CSE101",
        "name": "Introduction to Programming",
        "credits": 4,
        "type": "core",
        "subjects": [
          {
            "id": "uuid",
            "name": "C Programming",
            "credits": 3,
            "type": "theory"
          }
        ]
      }
    ],
    "pagination": {
      "currentPage": 1,
      "totalPages": 5,
```

```
    "totalCourses": 100
   }
  }
}
```

POST /api/v1/enrollments

Request:

```
{
  "courseId": "uuid",
  "academicYear": "2024",
  "semester": 1
}
```

Response (201):

```
{
  "success": true,
  "data": {
    "enrollmentId": "uuid",
    "courseId": "uuid",
    "status": "enrolled",
    "enrolledAt": "2024-12-15T10:00:00Z"
  },
  "message": "Successfully enrolled in course"
}
```

B.3 Assignment APIs

POST /api/v1/assignments

Request:

```
{
  "title": "Data Structures Assignment 1",
  "description": "Implement linked list operations",
```

```
  "courseId": "uuid",
  "subjectId": "uuid",
  "dueDate": "2024-12-25T23:59:59Z",
  "maxMarks": 100,
  "submissionType": "file",
  "allowLate": true,
  "latePenaltyPerDay": 5
}
```

Response (201):

```
{
  "success": true,
  "data": {
    "id": "uuid",
    "title": "Data Structures Assignment 1",
    "dueDate": "2024-12-25T23:59:59Z",
    "status": "draft"
  }
}
```

POST /api/v1/assignments/{id}/submit

Request (multipart/form-data):

```
{
  "files": [File],
  "comments": "Completed all requirements"
}
```

Response (201):

```
{
  "success": true,
  "data": {
    "submissionId": "uuid",
    "assignmentId": "uuid",
```

```
    "submittedAt": "2024-12-20T15:30:00Z",
    "isLate": false,
    "status": "submitted"
  },
  "message": "Assignment submitted successfully"
}
```

B.4 Content APIs

POST /api/v1/content/upload

Request:

```
{
  "title": "Lecture 1 - Introduction",
  "courseId": "uuid",
  "subjectId": "uuid",
  "type": "video",
  "fileName": "lecture1.mp4",
  "fileSize": 52428800
}
```

Response (200):

```
{
  "success": true,
  "data": {
    "uploadUrl": "https://s3.presigned.url",
    "contentId": "uuid",
    "expiresIn": 3600
  }
}
```

GET /api/v1/content/{id}/download

Response (200):

```
{
  "success": true,
  "data": {
    "downloadUrl": "https://s3.presigned.url",
    "expiresIn": 3600,
    "fileName": "lecture1.mp4"
  }
}
```

B.5 Chat APIs

Socket.IO Events:

```
// Client → Server
socket.emit('join_room', { roomId: 'course-123' });
socket.emit('send_message', {
  roomId: 'course-123',
  content: 'Hello everyone!',
  type: 'text'
});
socket.emit('typing', { roomId: 'course-123' });

// Server → Client
socket.on('receive_message', (message) => {
  // { id, roomId, senderId, senderName, content, timestamp }
});
socket.on('user_typing', (data) => {
  // { userId, userName }
});
```

B.6 Analytics APIs

GET /api/v1/analytics/students/{id}/dashboard

Response (200):


```
{
  "success": true,
  "data": {
    "gpa": 8.5,
    "attendancePercentage": 92.5,
    "enrolledCourses": 6,
    "completedAssignments": 15,
    "pendingAssignments": 3,
    "recentGrades": [
      {
        "assignmentTitle": "Assignment 1",
        "marks": 85,
        "maxMarks": 100,
        "gradedAt": "2024-12-10T10:00:00Z"
      }
    ]
  }
}
```

C. Kafka Topics Reference

C.1 Authentication Events

Topic	Event	Payload Schema
`auth.user.registered`	User registration	`{ userId, userType, email, name, timestamp }`

Topic	Event	Payload Schema
auth.user.login	User login	{ userId, userType, ipAddress, deviceId, timestamp }
auth.session.invalidated	Session invalidated	{ sessionId, userId, reason, timestamp }
auth.password.reset	Password reset request	{ userId, userType, email, resetToken, timestamp }
auth.audit.events	Audit log	{ userId, action, resource, outcome, timestamp }

C.2 Course Events

Topic	Event	Payload Schema
course.created	Course created	{ courseId, name, code, branchId, createdBy, timestamp }
course.updated	Course updated	{ courseId, changes, updatedBy, timestamp }
course.teacher.assigned	Teacher assigned	{ subjectId, teacherId, sectionId, timestamp }
enrollment.created	Student enrolled	{ enrollmentId, studentId, courseId, timestamp }
enrollment.section.allocated	Section allocated	{ enrollmentId, sectionId, timestamp }

C.3 Assignment Events

Topic	Event	Payload Schema
assignment.created		

Topic	Event	Payload Schema
	Assignment created	{ assignmentId, title, courseId, dueDate, timestamp }
assignment.published	Assignment published	{ assignmentId, publishedBy, timestamp }
submission.created	Submission received	{ submissionId, assignmentId, studentId, isLate, timestamp }
grade.released	Grade released	{ gradeId, submissionId, studentId, marks, timestamp }
assignment.deadline.approaching	Deadline reminder	{ assignmentId, studentId, hoursRemaining, timestamp }

C.4 Notification Events

Topic	Event	Payload Schema
notification.email.send	Send email	{ userId, template, data, timestamp }
notification.sms.send	Send SMS	{ userId, message, timestamp }
notification.push.send	Send push	{ userId, title, body, data, timestamp }
notification.account.locked	Account locked	{ userId, reason, timestamp }

C.5 Analytics Events

Topic	Event	Payload Schema
content.accessed	Content viewed	{ contentId, userId, duration, timestamp }

Topic	Event	Payload Schema
attendance.marked	Attendance marked	{ sessionId, studentId, status, timestamp }
chat.message.sent	Message sent	{ roomId, senderId, messageLength, timestamp }

D. Environment Variables

D.1 Auth Service

```
# Database
DATABASE_URL=postgresql://user:pass@localhost:5432/lms_auth
DATABASE_POOL_MIN=5
DATABASE_POOL_MAX=20

# Redis
REDIS_URL=redis://localhost:6379
REDIS_PASSWORD=
REDIS_DB=0

# Vault
VAULT_ADDR=http://localhost:8200
VAULT_TOKEN=root-token
VAULT_NAMESPACE=lms

# Kafka
KAFKA_BROKERS=localhost:9092
KAFKA_CLIENT_ID=auth-service
KAFKA_GROUP_ID=auth-service-group

# JWT/PASETO
```

```
JWT_SECRET=your-secret-key-here
PASETO_SECRET_KEY=your-paseto-key-here
ACCESS_TOKEN_EXPIRY=15m
REFRESH_TOKEN_EXPIRY=30d

# Email
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=noreply@lms.com
SMTP_PASSWORD=app-password

# Server
PORT=3000
NODE_ENV=development
LOG_LEVEL=debug
```

D.2 Course Service

```
DATABASE_URL=postgresql://user:pass@localhost:5432/lms_courses
REDIS_URL=redis://localhost:6379
KAFKA_BROKERS=localhost:9092
PORT=3001
```

D.3 Content Service

```
DATABASE_URL=postgresql://user:pass@localhost:5432/lms_content
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=your-key
AWS_SECRET_ACCESS_KEY=your-secret
S3_BUCKET_NAME=lms-content-dev
S3_PRESIGNED_URL_EXPIRY=3600
PORT=3002
```

D.4 Chat Service

```
MONGODB_URL=mongodb://localhost:27017/lms_chat
REDIS_URL=redis://localhost:6379
SOCKET_IO_PORT=3003
CORS_ORIGIN=http://localhost:3000
```

D.5 Infrastructure

```
# AWS
AWS_REGION=us-east-1
AWS_ACCOUNT_ID=123456789012
EKS_CLUSTER_NAME=lms-production
RDS_ENDPOINT=lms-db.xxx.rds.amazonaws.com
MSK_BOOTSTRAP_SERVERS=b-1.xxx.kafka.us-east-1.amazonaws.com:9092

# Monitoring
PROMETHEUS_URL=http://prometheus:9090
GRAFANA_URL=http://grafana:3000
ELASTICSEARCH_URL=http://elasticsearch:9200
```

E. Glossary

E.1 Technical Terms

Term	Definition
API Gateway	Entry point for all client requests, handles routing, authentication, rate limiting
Microservice	Independent, deployable service with single responsibility

Term	Definition
Event-Driven Architecture	System design where services communicate via events (Kafka)
PASETO	Platform-Agnostic Security Tokens, alternative to JWT
Idempotency	Property where multiple identical requests have same effect as single request
Circuit Breaker	Pattern to prevent cascading failures in distributed systems
Service Mesh	Infrastructure layer for service-to-service communication
Horizontal Scaling	Adding more instances of a service to handle load
Vertical Scaling	Increasing resources (CPU, RAM) of existing instance
Blue-Green Deployment	Deployment strategy with two identical environments
Canary Deployment	Gradual rollout to subset of users

E.2 Academic Terms

Term	Definition
Branch	Department or specialization (e.g., Computer Science, Mechanical)
Batch	Group of students admitted in same year
Section	Division of batch for classroom management (e.g., Section A, B)
Semester	Half of academic year (typically 6 months)
Credits	Units representing course workload
GPA	Grade Point Average, measure of academic performance
Prerequisite	Course that must be completed before enrolling in another
Elective	Optional course student can choose

Term	Definition
Core Course	Mandatory course for degree completion
Syllabus	Outline of topics covered in course
Regularization	Process to correct attendance/grade records

E.3 Acronyms

Acronym	Full Form
LMS	Learning Management System
API	Application Programming Interface
REST	Representational State Transfer
CRUD	Create, Read, Update, Delete
JWT	JSON Web Token
PASETO	Platform-Agnostic Security Tokens
OTP	One-Time Password
2FA	Two-Factor Authentication
RBAC	Role-Based Access Control
SSO	Single Sign-On
SMTP	Simple Mail Transfer Protocol
S3	Simple Storage Service (AWS)
RDS	Relational Database Service (AWS)
EKS	Elastic Kubernetes Service (AWS)
MSK	Managed Streaming for Kafka (AWS)

Acronym	Full Form
VPC	Virtual Private Cloud
ALB	Application Load Balancer
NAT	Network Address Translation
CIDR	Classless Inter-Domain Routing
TLS	Transport Layer Security
CORS	Cross-Origin Resource Sharing
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
WCAG	Web Content Accessibility Guidelines
SLA	Service Level Agreement
RPO	Recovery Point Objective
RTO	Recovery Time Objective

F. Migration Guide

F.1 From Existing LMS

Pre-Migration Checklist:

1. ✅ Audit current data (users, courses, grades)
2. ✅ Identify data quality issues
3. ✅ Map old schema to new schema
4. ✅ Plan downtime window
5. ✅ Set up rollback plan
6. ✅ Communicate with stakeholders

Migration Steps:

1. **Export Data from Old System**
    - Users (students, teachers, admins)
    - Courses and subjects
    - Enrollments
    - Grades and transcripts
    - Content files
2. **Data Transformation**
    - Clean and validate data
    - Transform to new schema format
    - Generate UUIDs for new system
    - Hash passwords (if migrating)
3. **Import to New System**
    - Use bulk import APIs
    - Import in order: Organizations → Colleges → Branches → Batches → Users → Courses → Enrollments
    - Validate each step
4. **Content Migration**
    - Upload files to S3
    - Update database with S3 URLs
    - Verify file accessibility
5. **Post-Migration Validation**
    - User count matches
    - Course enrollments correct
    - Grades preserved
    - Run test scenarios

F.2 Data Migration Steps

Step 1: Export Users

```
-- Old system query
SELECT id, name, email, role, created_at
FROM users
WHERE is_active = true;
```

Step 2: Transform to CSV

```
name,email,role,password,collegeId,branchId
John Doe,john@example.com,student,TempPass123!,uuid-college,uuid-branch
```

Step 3: Bulk Import

```
curl -X POST http://api.lms.com/api/v1/bulk/students/import \
  -H "Authorization: Bearer $TOKEN" \
  -F "file=@students.csv"
```

F.3 User Migration

Password Reset Flow:

1. Import users with temporary passwords
2. Send password reset emails to all users
3. Users set new passwords on first login
4. Enforce password policy

Session Migration:

- Old sessions invalidated
- Users must re-login to new system
- Communicate downtime in advance

G. Security Best Practices

G.1 Authentication Security

1. **Password Storage**
    - Use Argon2id for hashing
    - Salt automatically generated
    - Never store plaintext passwords
2. **Token Management**
    - Short-lived access tokens (15 minutes)
    - Refresh token rotation
    - Secure token storage (httpOnly cookies)
3. **Session Security**
    - Single device login enforcement
    - Session timeout after inactivity
    - Invalidate on password change

G.2 API Security

1. **Rate Limiting**
    - Prevent brute-force attacks
    - Per-IP and per-user limits
    - Exponential backoff
2. **Input Validation**
    - Validate all inputs (Zod/Joi)
    - Sanitize user content
    - Prevent SQL injection
3. **CORS Configuration**
    - Whitelist allowed origins
    - No wildcard in production

- Credentials allowed only for trusted origins

G.3 Data Protection

1. **Encryption**
    - TLS 1.3 for data in transit
    - Encrypt sensitive data at rest
    - Use AWS KMS for key management
2. **Access Control**
    - Principle of least privilege
    - Role-based access control
    - Audit all access to sensitive data
3. **Data Retention**
    - Define retention policies
    - Automated deletion of old data
    - Comply with GDPR right to deletion

G.4 Infrastructure Security

1. **Network Security**
    - Private subnets for databases
    - Security groups (restrictive rules)
    - VPC endpoints for AWS services
2. **Secrets Management**
    - Use Vault or AWS Secrets Manager
    - Rotate secrets regularly
    - Never commit secrets to Git
3. **Monitoring**
    - Real-time security alerts
    - Audit log analysis
    - Intrusion detection

H. Performance Benchmarks

H.1 API Response Times (p95)

Endpoint	Target	Acceptable	Critical
GET /courses	<200ms	<500ms	>1s
POST /login	<300ms	<800ms	>2s
POST /assignments/{id}/submit	<2s	<5s	>10s
GET /students/{id}/dashboard	<500ms	<1s	>3s
WebSocket message delivery	<100ms	<300ms	>1s

H.2 Database Query Performance

Query Type	Target	Optimization
Simple SELECT	<10ms	Indexed columns
JOIN (2 tables)	<50ms	Proper indexes
Complex aggregation	<200ms	Materialized views
Full-text search	<100ms	Elasticsearch

H.3 Frontend Load Times

Metric	Target	Tool
First Contentful Paint	<1.5s	Lighthouse
Time to Interactive	<3s	Lighthouse
Largest Contentful Paint	<2.5s	Lighthouse

Metric	Target	Tool
Cumulative Layout Shift	<0.1	Lighthouse

H.4 Scalability Metrics

Metric	Current	Target (1 year)
Concurrent users	1,000	10,000
Requests per second	500	5,000
Database size	10 GB	100 GB
S3 storage	100 GB	1 TB
Kafka messages/day	100K	1M

End of Appendix

This appendix provides comprehensive reference materials for the LMS Platform. For implementation details, refer to the phase-specific documents.

Last Updated: December 15, 2025
Maintained By: Technical Architecture Team

LMS Platform - Complete Phase Breakdown (Phases 6-25)

Phase 6: Course Management Service

Goals: Create, read, update, delete courses; manage curriculum; assign teachers to courses

Key Deliverables:

- New microservice: `course-service`
- Course CRUD APIs
- Course-subject-teacher relationships
- Curriculum versioning
- Course prerequisites logic

Folder Structure Additions:

```
services/course-service/
├── src/
│   ├── controllers/course.controller.ts
│   ├── services/course.service.ts
│   ├── models/ (Prisma schema)
│   └── routes/course.route.ts
```

Verification: Course creation, teacher assignment, prerequisite validation

---

Phase 7: Student Enrollment & Allocation

Goals: Enroll students in courses; auto-allocate to sections; manage waitlists

Key Deliverables:

- Enrollment APIs
- Section capacity management
- Auto-allocation algorithm
- Waitlist functionality
- Enrollment notifications

Folder Structure Additions:

```
services/course-service/
├── src/
│   ├── controllers/enrollment.controller.ts
│   ├── services/enrollment.service.ts
│   └── utils/allocation-algorithm.ts
```

Verification: Bulk enrollment, capacity limits, waitlist processing

---

Phase 8: Content Management System (CMS)

Goals: Upload, store, and deliver course content (videos, PDFs, presentations)

Key Deliverables:

- S3 integration for file storage
- Presigned URL generation
- Content versioning
- Access control (role-based)
- Video transcoding (optional)

Folder Structure Additions:

```
services/content-service/
├── src/
│   ├── controllers/content.controller.ts
│   ├── services/
```

```
│   │   ├── s3.service.ts
│   │   ├── upload.service.ts
│   │   └── transcoding.service.ts (optional)
│   └── middleware/fileUpload.ts
```

Verification: File upload (50MB), presigned URL access, access control

---

Phase 9: Attendance Service

Goals: Mark attendance; geofencing; QR code-based attendance; reports

Key Deliverables:

- Attendance marking APIs
- Geofencing logic (optional)
- QR code generation/scanning
- Attendance reports
- Attendance analytics

Folder Structure Additions:

```
services/attendance-service/
├── src/
│   ├── controllers/attendance.controller.ts
│   ├── services/
│   │   ├── attendance.service.ts
│   │   ├── geofencing.service.ts
│   │   └── qr.service.ts
│   └── utils/attendance-calculator.ts
```

Verification: Attendance marking, geofence validation, report generation

---

Phase 10: Assignment & Submission Service

Goals: Create assignments; student submissions; file uploads; plagiarism check (basic)

Key Deliverables:

- Assignment CRUD APIs
- Submission workflow
- File upload for submissions
- Deadline enforcement
- Submission notifications

Folder Structure Additions:

```
services/assignment-service/
├── src/
│   ├── controllers/
│   │   ├── assignment.controller.ts
│   │   └── submission.controller.ts
│   ├── services/
│   │   ├── assignment.service.ts
│   │   ├── submission.service.ts
│   │   └── plagiarism.service.ts (basic)
│   └── utils/deadline-checker.ts
```

Verification: Assignment creation, submission upload, deadline validation

---

Phase 11: Grading & Assessment Engine

Goals: Grade submissions; rubrics; auto-grading (MCQ); grade analytics

Key Deliverables:

- Grading APIs
- Rubric management
- Auto-grading for MCQs

- Grade calculation (weighted average)
- Grade distribution analytics

Folder Structure Additions:

```
services/assignment-service/
├── src/
│   ├── controllers/grading.controller.ts
│   ├── services/
│   │   ├── grading.service.ts
│   │   ├── rubric.service.ts
│   │   └── auto-grading.service.ts
│   └── utils/grade-calculator.ts
```

Verification: Manual grading, auto-grading MCQ, grade calculation

---

Phase 12: Real-time Chat Service

Goals: One-on-one and group chat; file sharing; message history

Key Deliverables:

- Socket.IO integration
- Chat rooms (course-based, direct)
- Message persistence (MongoDB)
- File sharing in chat
- Typing indicators, read receipts

Folder Structure Additions:

```
services/chat-service/
├── src/
│   ├── socket/
│   │   ├── chat.handler.ts
│   │   └── room.handler.ts
│   ├── services/
```

```
│   │   ├── chat.service.ts
│   │   └── message.service.ts
│   ├── models/ (MongoDB schemas)
│   └── middleware/socket-auth.ts
```

Verification: Real-time messaging, file sharing, message persistence

---

Phase 13: Search Service (Elasticsearch)

Goals: Full-text search for courses, users, content; filters; autocomplete

Key Deliverables:

- Elasticsearch integration
- Indexing pipeline (Kafka → ES)
- Search APIs with filters
- Autocomplete/suggestions
- Search analytics

Folder Structure Additions:

```
services/search-service/
├── src/
│   ├── controllers/search.controller.ts
│   ├── services/
│   │   ├── elasticsearch.service.ts
│   │   ├── indexer.service.ts
│   │   └── search.service.ts
│   └── workers/index-sync.worker.ts
```

Verification: Search accuracy, autocomplete, filter combinations

---

Phase 14: Analytics & Reporting Service

Goals: Student performance analytics; course analytics; custom reports; dashboards

Key Deliverables:

- Analytics APIs
- Pre-built reports (attendance, grades, enrollment)
- Custom report builder
- Data export (CSV, PDF)
- Dashboard widgets

Folder Structure Additions:

```
services/analytics-service/
├── src/
│   ├── controllers/
│   │   ├── analytics.controller.ts
│   │   └── report.controller.ts
│   ├── services/
│   │   ├── analytics.service.ts
│   │   ├── report-generator.service.ts
│   │   └── export.service.ts
│   └── templates/ (report templates)
```

Verification: Report generation, data accuracy, export formats

---

Phase 15: Notification Enhancement

Goals: SMS notifications; push notifications (web/mobile); notification preferences

Key Deliverables:

- SMS integration (Twilio)
- Web push notifications
- Mobile push (FCM)

- Notification preferences API
- Notification history

Folder Structure Additions:

```
services/notification-service/
├── src/
│   ├── services/
│   │   ├── sms.service.ts
│   │   ├── push.service.ts
│   │   └── preference.service.ts
│   ├── handlers/
│   │   ├── sms.handler.ts
│   │   └── push.handler.ts
│   └── templates/ (SMS templates)
```

Verification: SMS delivery, push notification, preference management

---

Phase 16: Timetable & Scheduling Service

Goals: Create timetables; room allocation; conflict detection; calendar integration

Key Deliverables:

- Timetable CRUD APIs
- Room/resource booking
- Conflict detection algorithm
- iCal export
- Calendar view APIs

Folder Structure Additions:

```
services/timetable-service/
├── src/
│   ├── controllers/
│   │   ├── timetable.controller.ts
```

```
│   │   └── room.controller.ts
│   ├── services/
│   │   ├── timetable.service.ts
│   │   ├── scheduling.service.ts
│   │   └── conflict-detector.service.ts
│   └── utils/ical-generator.ts
```

Verification: Timetable creation, conflict detection, iCal export

---

Phase 17: Library Management Integration

Goals: Library catalog; book issue/return; fine management; integration with existing library systems

Key Deliverables:

- Library catalog APIs
- Issue/return workflow
- Fine calculation
- Integration with library management systems (optional)
- Book search

Folder Structure Additions:

```
services/library-service/
├── src/
│   ├── controllers/
│   │   ├── catalog.controller.ts
│   │   └── transaction.controller.ts
│   ├── services/
│   │   ├── catalog.service.ts
│   │   ├── transaction.service.ts
│   │   └── fine.service.ts
│   └── integrations/ (external library systems)
```

Verification: Book issue, return, fine calculation

Phase 18: Payment & Fee Management

Goals: Fee structure; payment gateway integration; receipts; payment history

Key Deliverables:

- Fee structure APIs
- Payment gateway integration (Stripe/Razorpay)
- Receipt generation
- Payment history
- Refund workflow

Folder Structure Additions:

```
services/payment-service/
├── src/
│   ├── controllers/
│   │   ├── fee.controller.ts
│   │   └── payment.controller.ts
│   ├── services/
│   │   ├── payment-gateway.service.ts
│   │   ├── receipt.service.ts
│   │   └── refund.service.ts
│   └── integrations/
│       ├── stripe.ts
│       └── razorpay.ts
```

Verification: Payment processing, receipt generation, refund

Phase 19: Super Admin Portal (Frontend)

Goals: Next.js portal for organization admins; manage colleges, users, system settings

Key Deliverables:

- Next.js application
- Organization dashboard
- College management UI
- User management UI
- System settings UI
- Responsive design

Folder Structure Additions:

```
frontend/super-admin-portal/
├── src/
│   ├── app/ (Next.js 14 app router)
│   ├── components/
│   ├── lib/ (API clients)
│   ├── hooks/
│   └── styles/
├── public/
└── package.json
```

Verification: UI functionality, responsiveness, API integration

---

Phase 20: College Management Portal (Frontend)

Goals: Next.js portal for college admins, deans, HODs; manage courses, faculty, students

Key Deliverables:

- Next.js application
- College dashboard
- Course management UI
- Faculty management UI
- Student management UI

- Reports UI

Folder Structure Additions:

```
frontend/college-portal/
├── src/
│   ├── app/
│   ├── components/
│   ├── lib/
│   ├── hooks/
│   └── styles/
├── public/
└── package.json
```

Verification: UI functionality, role-based access, reports

Phase 21: Student LMS Portal (Frontend)

Goals: Next.js portal for students; course access, assignments, grades, chat

Key Deliverables:

- Next.js application
- Student dashboard
- Course catalog and enrollment
- Assignment submission UI
- Grades view
- Chat interface
- Attendance view

Folder Structure Additions:

```
frontend/student-portal/
├── src/
│   ├── app/
│   ├── components/
```

```
│   ├── lib/
│   ├── hooks/
│   └── styles/
├── public/
└── package.json
```

Verification: UI functionality, real-time features, mobile responsiveness

---

Phase 22: AWS Infrastructure (Terraform)

Goals: Production-ready AWS infrastructure; EKS/ECS, RDS, MSK, S3, CloudWatch

Key Deliverables:

- Terraform modules
- VPC setup
- EKS cluster (or ECS)
- RDS PostgreSQL
- MSK (Kafka)
- S3 buckets
- Elasticache (Redis)
- CloudWatch logging
- IAM roles/policies

Folder Structure Additions:

```
infrastructure/
├── terraform/
│   ├── modules/
│   │   ├── vpc/
│   │   ├── eks/
│   │   ├── rds/
│   │   ├── msk/
│   │   ├── s3/
│   │   └── elasticache/
```

```
│   ├── environments/
│   │   ├── dev/
│   │   ├── staging/
│   │   └── production/
│   └── main.tf
└── kubernetes/
    ├── deployments/
    ├── services/
    └── ingress/
```

Verification: Infrastructure provisioning, service deployment, connectivity

---

Phase 23: CI/CD Pipeline (GitHub Actions)

Goals: Automated build, test, deploy; multi-environment; rollback capability

Key Deliverables:

- GitHub Actions workflows
- Docker image builds
- Automated testing (unit, integration, E2E)
- Deployment to EKS/ECS
- Rollback mechanism
- Slack notifications

Folder Structure Additions:

```
.github/
├── workflows/
│   ├── build-and-test.yml
│   ├── deploy-dev.yml
│   ├── deploy-staging.yml
│   ├── deploy-production.yml
│   └── rollback.yml
└── actions/ (custom actions)
```

Verification: Successful builds, automated tests, deployments

---

Phase 24: Mobile Applications (React Native)

Goals: iOS and Android apps for students and teachers; offline support

Key Deliverables:

- React Native application
- Student features (courses, assignments, chat)
- Teacher features (attendance, grading)
- Push notifications
- Offline mode (basic)
- App store deployment

Folder Structure Additions:

```
mobile/
├── src/
│   ├── screens/
│   ├── components/
│   ├── navigation/
│   ├── services/ (API clients)
│   └── utils/
├── android/
├── ios/
└── package.json
```

Verification: App functionality, offline mode, push notifications

---

Phase 25: AI/ML Features

Goals: Personalized recommendations; auto-grading essays; proctoring; chatbot

Key Deliverables:

- Course recommendation engine
- AI-powered essay grading
- Online exam proctoring (face detection, tab switching)
- AI chatbot for student queries
- Predictive analytics (at-risk students)

Folder Structure Additions:

```
services/ai-service/
├── src/
│   ├── controllers/
│   │   ├── recommendation.controller.ts
│   │   ├── proctoring.controller.ts
│   │   └── chatbot.controller.ts
│   ├── services/
│   │   ├── recommendation.service.ts
│   │   ├── essay-grading.service.ts
│   │   ├── proctoring.service.ts
│   │   └── chatbot.service.ts
│   └── models/ (ML models)
└── ml/
    ├── notebooks/ (Jupyter)
    ├── datasets/
    └── trained-models/
```

Verification: Recommendation accuracy, proctoring detection, chatbot responses

---

Implementation Timeline

Year 1 (Phases 1-11): MVP & Core Features

- **Q1**: Phases 1-5 (Foundation & Security)
- **Q2**: Phases 6-8 (Course Management, Enrollment, CMS)

- **Q3**: Phases 9-11 (Attendance, Assignments, Grading)
- **Q4**: Testing, bug fixes, internal alpha

Year 2 (Phases 12-21): Advanced Features & Frontend

- **Q1**: Phases 12-15 (Chat, Search, Analytics, Notifications)
- **Q2**: Phases 16-18 (Timetable, Library, Payments)
- **Q3**: Phases 19-20 (Admin & College Portals)
- **Q4**: Phase 21 (Student Portal), beta testing

Year 3 (Phases 22-25): Production & Scale

- **Q1**: Phase 22 (AWS Infrastructure)
- **Q2**: Phase 23 (CI/CD), production deployment
- **Q3**: Phase 24 (Mobile Apps)
- **Q4**: Phase 25 (AI/ML), general availability

For detailed requirements, user stories, technical architecture, and verification steps for each phase, refer to the individual phase documents (01-PHASES-01-05.md, etc.).

LMS Platform - Product Requirements Document (PRD)

Document Overview

This PRD provides a comprehensive roadmap for developing the LMS Platform across 25 phases. The documentation is split into multiple files for manageability.

Document Structure

Core Documents

- **00-OVERVIEW.md** - Project overview, goals, metrics, tech stack, current state
- **01-PHASES-01-05.md** - Foundation & Security (Auth hardening, single device login, bulk imports, monitoring, API gateway)
- **02-PHASES-06-10.md** - Core LMS Part 1 (Course management, enrollment, CMS, attendance, assignments)
- **03-PHASES-11-15.md** - Core LMS Part 2 & Advanced (Grading, chat, search, analytics, notifications)
- **04-PHASES-16-20.md** - Advanced Features & Frontend (Timetable, library, payments, admin portal, college portal)
- **05-PHASES-21-25.md** - Production & Scale (Student portal, AWS infrastructure, CI/CD, mobile apps, AI/ML)
- **06-APPENDIX.md** - Data models, API contracts, glossary, migration guide

Quick Reference

Phase Summary

Phase	Name	Status	Priority
1	Auth Service Hardening	In Progress	P0
2	Single Device Login	In Progress	P0
3	Bulk Import Infrastructure	⚫ Pending	P0
4	Monitoring & Observability	⚫ Pending	P0
5	API Gateway Enhancement	⚫ Pending	P1
6	Course Management Service	⚫ Pending	P0
7	Student Enrollment & Allocation	⚫ Pending	P0
8	Content Management System	⚫ Pending	P0
9	Attendance Service	⚫ Pending	P1
10	Assignment & Submission	⚫ Pending	P0
11	Grading & Assessment	⚫ Pending	P0
12	Real-time Chat Service	⚫ Pending	P2
13	Search Service (Elasticsearch)	⚫ Pending	P1
14	Analytics & Reporting	⚫ Pending	P1
15	Notification Enhancement	⚫ Pending	P2
16	Timetable & Scheduling	⚫ Pending	P1
17	Library Management	⚫ Pending	P2
18	Payment & Fee Management	⚫ Pending	P2

Phase	Name	Status	Priority
19	Super Admin Portal	🔘 Pending	P0
20	College Management Portal	🔘 Pending	P0
21	Student LMS Portal	🔘 Pending	P0
22	AWS Infrastructure (Terraform)	🔘 Pending	P1
23	CI/CD Pipeline	🔘 Pending	P1
24	Mobile Applications	🔘 Pending	P2
25	AI/ML Features	🔘 Pending	P3

Legend

- ✅ Completed
- 🔄 In Progress
- 🔘 Pending
- **P0**: Critical (MVP)
- **P1**: High Priority
- **P2**: Medium Priority
- **P3**: Future Enhancement

How to Use This PRD

For Product Managers

1. Start with **00-OVERVIEW.md** for project goals and success metrics
2. Review phase summaries to understand feature roadmap
3. Use user stories for backlog creation
4. Track verification steps for release criteria

For Engineers

1. Read technical architecture sections for each phase

2. Review folder/file structure updates
3. Check dependencies and tech stack requirements
4. Use verification steps for testing guidance

For QA/Testing

1. Focus on verification steps at end of each phase
2. Use acceptance criteria from user stories
3. Reference load testing scenarios
4. Check non-functional requirements

For DevOps

1. Review infrastructure requirements (Phase 4, 22, 23)
2. Check monitoring and observability setup (Phase 4)
3. Review deployment architecture (Phase 22)
4. Plan capacity based on scalability requirements

Current Implementation Status

✅ Completed Components

- Edge authentication with Kong Gateway + PASETO + Vault
- Multi-role authentication (7 user types)
- Database schema with academic hierarchy
- Kafka messaging infrastructure
- Email notification service
- Docker Compose development environment

Partially Completed

- Single device login (schema exists, enforcement pending)
- Bulk data import (documentation exists, async processing pending)
- API documentation (Swagger partially configured)

⚪ Not Started

- Frontend applications
- Core LMS services (courses, attendance, grading)
- Monitoring stack (Prometheus, Grafana, ELK)
- Cloud deployment (AWS)
- Advanced features (analytics, AI/ML)

Development Workflow

Phase Execution Process

1. **Planning** (Week 1)
    - Review phase goals and user stories
    - Estimate effort and assign tasks
    - Set up development environment
2. **Development** (Weeks 2-4)
    - Implement functional requirements
    - Write unit tests (>80% coverage)
    - Update documentation
3. **Testing** (Week 5)
    - Execute verification steps
    - Perform load testing
    - Security testing (if applicable)
4. **Review & Deploy** (Week 6)
    - Code review
    - Deploy to staging
    - Stakeholder demo
    - Deploy to production (if approved)

Typical Phase Duration

- **Small phases** (1-5, 13-15): 4-6 weeks
- **Medium phases** (6-12, 16-18): 6-8 weeks
- **Large phases** (19-21, 22-25): 8-12 weeks

Key Architectural Decisions

Microservices Boundaries

- **Auth Service**: User management, authentication, authorization
- **Course Service**: Course CRUD, curriculum management
- **Content Service**: Media storage, document management
- **Attendance Service**: Attendance tracking, geofencing
- **Assignment Service**: Assignment creation, submission, grading
- **Notification Service**: Email, SMS, push notifications
- **Analytics Service**: Reporting, dashboards, insights
- **Chat Service**: Real-time messaging, collaboration

Data Ownership

- **PostgreSQL**: Structured data (users, courses, enrollments, grades)
- **MongoDB**: Unstructured content (notes, chat history)
- **S3**: Media files (videos, documents, images)
- **Redis**: Cache, sessions, real-time data
- **Elasticsearch**: Search indexes, logs

Communication Patterns

- **Synchronous**: REST APIs for CRUD operations
- **Asynchronous**: Kafka for events, notifications, background jobs
- **Real-time**: Socket.IO for chat, live updates

Success Criteria

MVP (Phases 1-11)

- ✅ Secure authentication for all user types
- ✅ Course creation and enrollment
- ✅ Content delivery (documents, videos)
- ✅ Assignment submission and grading
- ✅ Attendance tracking
- ✅ Basic reporting
- ✅ 99% uptime
- ✅ <500ms API response time

Beta Release (Phases 12-18)

- ✅ Real-time chat and collaboration
- ✅ Advanced search
- ✅ Analytics dashboards
- ✅ Timetable management
- ✅ Payment integration
- ✅ 5,000+ active users
- ✅ 99.5% uptime

General Availability (Phases 19-25)

- ✅ Full-featured web portals
- ✅ Mobile applications
- ✅ AI-powered features
- ✅ Cloud-native deployment
- ✅ 50,000+ users
- ✅ 99.9% uptime

Contact & Support

Product Owner: [Name]
Technical Lead: [Name]
Project Manager: [Name]

Slack Channels:

- #lms-product - Product discussions
- #lms-engineering - Technical discussions
- #lms-qa - Testing and quality
- #lms-alerts - Production alerts

Documentation:

- Wiki: https://wiki.example.com/lms
- API Docs: https://api.lms.example.com/docs
- Runbooks: https://wiki.example.com/lms/runbooks

Document Version: 1.0
Last Updated: December 2025
Next Review: January 2026