Requirements Specification Document

1. Introduction

This document defines the functional and non-functional requirements, user stories, use cases, and architectural drivers for an online learning platform (LMS) based on a microservices architecture.

---

2. Functional Requirements

User Authentication & Authorization

- Users can register and log in via Google OAuth2.
- Role-based access control (Admin, Instructor, Student).
- Users can update their profile (name, email, profile picture).

Course Management

- Instructors can create, update, delete, and publish courses.
- Students can browse, enroll, add,and drop courses.
- Course materials (videos, PDFs, quizzes) can be uploaded and accessed.
- Support for course prerequisites (students must complete Course A before Course B).
- Instructors can schedule live sessions (Zoom/Google Meet integration).

Enrollment & Payment

- Users can enroll in free or paid courses.
- Automatic invoicing and receipt generation.
- Support for refunds and cancellations.

Recommendation System

- AI-based personalized course recommendations based on user behavior.

Notification System

-Real-time notifications for:

- Enrollment confirmation
- Course updates
- Assignment deadlines
- Payment confirmations

-Support for email, in-app, and push notifications.

Dashboard & Analytics

-Students can track course progress (completion percentage, quiz scores).
-Instructors can view student engagement metrics.
- Admins can monitor platform analytics (user growth, course popularity, system health).

---

3. Non-Functional Requirements

Performance

- The API must handle 10,000 requests per second (RPS) at peak load.
- The database should support 20,000 queries per second (QPS) using read replicas.
- Video streaming should support 2,500 concurrent users with low latency.
- The system should provide a response time of less than 500ms for 95% of requests.

Scalability

- The system should support 100,000 registered users with 5,000 concurrent users.
- Auto-scaling must be implemented for API and database services to handle traffic spikes.
- The system should allow dynamic resource provisioning based on traffic load.
- A Content Delivery Network (CDN) should be used to optimize static content delivery.

Availability

- The system must ensure 99.9% uptime, limiting downtime to 8.76 hours/year.
- The application should support graceful degradation, ensuring core functionalities remain operational during failures.
- Redundant instances of critical services should be deployed to prevent single points of failure.

4. Resilience

- Circuit breakers & retry mechanisms should be implemented to handle failures in service-to-service communication.
- Fallback strategies should be designed for critical services to prevent complete system failure.
- A distributed message queue (e.g., Kafka, RabbitMQ) should be used for asynchronous processing (e.g., notifications, payments).
- Database replication and sharding should ensure failover support and high availability.

Security

- OAuth2-based authentication using Google Login must be implemented for secure access.
- Role-based access control (RBAC) must restrict functionalities for Admins, Instructors, and Students.
- Sensitive user data (e.g., passwords, payment details) must be encrypted at rest and in transit using AES-256 and TLS 1.2+.
- The system should comply with GDPR and ISO 27001 standards for data protection.
- Web Application Firewall (WAF) should be used to prevent security threats such as DDoS attacks and SQL injection.

Maintainability

- Centralized logging using OpenSearch should be implemented for debugging and security monitoring.
- Microservices should be loosely coupled to allow independent updates and deployments.
- The system must follow a modular architecture, allowing easy replacement or enhancement of components.

Observability

- Application metrics (e.g., API latency, database queries, active users) should be collected using Prometheus/Grafana.
- Distributed tracing should be implemented to track requests across microservices (e.g., Jaeger, Zipkin).
- Alerting mechanisms should be in place to notify administrators of critical failures.

Deployment & CI/CD

- Automated CI/CD pipelines should be used for deployments to minimize manual errors.
- Rolling updates should be used to prevent downtime during deployment.
- Blue-Green Deployment should be considered for major releases to ensure rollback capability.

Storage Requirements

- Each course requires an average of 5GB of storage (videos, documents).
- With 10,000 courses, total storage required = 50TB, hosted on cloud storage (e.g., AWS S3, GCS).
- 90% of video traffic should be served via CDN, reducing direct storage bandwidth.
- Database backup and recovery policies should ensure data protection and minimize data loss.

---

4. User Stories

As an Admin, I want to:

- Manage users (approve instructors, suspend accounts) so that I can ensure platform security.
-Monitor platform usage (active users, enrollments, payments) so that I can maintain system performance.
- View reports and analytics so that I can track student engagement and instructor activity.
- Manage system settings (payment configurations, email notifications) so that I can customize platform behavior.

- Moderate content (course reviews, discussion forums) so that I can prevent spam or inappropriate content.

As an Instructor, I want to:

- Create, edit, and delete courses so that I can provide learning content to students.
- Upload course materials (videos, PDFs, quizzes) so that students can access structured learning resources.
- Schedule live sessions so that I can interact with students in real time.
- Track student progress so that I can provide personalized support.
- Send announcements so that I can notify students about course updates.
- View student engagement metrics (time spent, quiz scores) so that I can improve course effectiveness.

As a Student, I want to:

- Search and browse courses so that I can find relevant learning opportunities.
- Enroll in a course (free or paid) so that I can access learning materials.
- Pay for premium courses so that I can unlock additional content.
- Receive personalized course recommendations so that I can discover relevant learning paths.
- Track my progress (completed lessons, quiz scores) so that I can measure my learning.
- Receive notifications (new lessons, deadlines, announcements) so that I stay updated.
- Engage in course discussions so that I can ask and answer questions.
- Download certificates after course completion so that I can showcase my learning achievements.

As a System (LMS), I want to:

- Authenticate users via Google OAuth2 so that login is secure and seamless.
- Store user roles and permissions so that access control is enforced.
- Process payments securely so that transactions are safe.
- Send automated notifications so that users are informed about updates.
- Analyze user behavior so that course recommendations are personalized.
- Ensure high availability (scalability and resilience) so that the system runs smoothly under heavy load.

5. Use Cases

Use Case 1: User Registration & Authentication

- **Use Case ID:** UC-01
- **Actors:** Student, Instructor, Admin
- **Preconditions:** User must have a Google account.
- **Basic Flow:**
  - The user clicks the "Sign Up/Login" button.
  - The system redirects the user to Google OAuth2 for authentication.
  - The user selects a Google account to sign in.
  - The system verifies the user’s identity and creates a session.
  - The system assigns a role (Admin, Instructor, or Student) based on user data.
  - The user is redirected to the respective dashboard.
- **Alternative Flow:**
  - **A1:** If authentication fails, the system displays an error message and allows the user to retry.
- **Postconditions:** The user is authenticated and redirected to their dashboard.

Use Case 2: Course Creation

- **Use Case ID:** UC-02
- **Actors:** Instructor

- **Preconditions:** The instructor must be logged in.
- **Basic Flow:**
    - The instructor clicks the "Create Course" button.
    - The system displays a course creation form.
    - The instructor enters the course details (title, description, etc.).
    - The system validates the input.
    - The instructor uploads course materials (videos, documents).
    - The system stores the files and links them to the course.
    - The instructor clicks "Publish."
    - The system makes the course visible for enrollment.
- **Alternative Flow:**
    - **A1:** If validation fails, the system prompts the instructor to correct errors before proceeding.
    - **A2:** If file upload fails, the system notifies the instructor to retry.
- **Postconditions:** The course is available for enrollment.

Use Case 3: Course Enrollment

- **Use Case ID:** UC-03
- **Actors:** Student
- **Preconditions:** The student must be logged in.

- **Basic Flow:**
    - The student searches for a course.
    - The system displays a list of available courses.
    - The student clicks "Enroll" for a selected course.
    - The system checks if the course is free or requires payment.
    - If the course is paid, the student completes the payment process.
    - The system processes the payment and confirms enrollment.
    - The student receives a confirmation notification.
    - The course is added to the student's dashboard.
- **Alternative Flow:**
    - **A1:** If payment fails, the system notifies the user and provides retry options.
    - **A2:** If course capacity is full, the system displays a waitlist option.
- **Postconditions:** The student gains access to the course.

Use Case 4: Payment Processing

- **Use Case ID:** UC-04
- **Actors:** Student
- **Preconditions:** The student selects a paid course.
- **Basic Flow:**

    - The student selects a premium (paid) course.
    - The system redirects the student to the payment page.
    - The student enters payment details.
    - The system validates and processes the transaction.
    - The student confirms the payment.
    - The system adds the course to the student's dashboard.
    - The system records the transaction in the database.
    - The student receives a payment confirmation email.
- **Alternative Flow:**
    - **A1:** If payment is declined, the system notifies the user and prompts for alternative payment methods.
    - **A2:** If the transaction times out, the system provides an option to retry.
- **Postconditions:** The student is enrolled, and the payment is recorded.

---

6. Architectural Drivers (Quality Attributes)

Scalability

The system ensures scalability by using a Content Delivery Network (CDN) to optimize content delivery, auto-scaling mechanisms to dynamically adjust resources based on demand, and load balancing to distribute traffic efficiently across multiple instances.

Resilience

To maintain system reliability, the application implements retry mechanisms to handle transient failures, circuit breakers to prevent cascading failures, and caching to reduce load on backend services and improve response times.

Security

Security is enforced through OAuth2-based authentication for secure user login, Role-Based Access Control (RBAC) to restrict access based on user roles, encryption for data protection, and rate limiting to mitigate potential abuse or DDoS attacks.

Performance

The system optimizes performance by utilizing caching to reduce redundant database queries, asynchronous processing for background tasks (e.g., notifications, report generation), and optimized database queries to enhance data retrieval efficiency.

Observability

For monitoring and observability, the application integrates OpenSearch for centralized logging, Prometheus for collecting system metrics, and Grafana for real-time visualization and alerting to detect potential issues proactively.