= SPEC-001: E-Courses Platform :sectnums: :toc:

== Background

This design addresses the growing need for flexible digital learning platforms that support both recorded and live educational content. The system serves students, instructors, and administrators. It is built as a Modular Monolith to simplify development while keeping a clean separation of concerns and potential for scaling into microservices. Initial focus is on a web-based backend system.

== Requirements

=== Must Have

- Secure user authentication with OTP (email) and JWT
- Role-based access (student, instructor, admin)
- Admin/instructor-controlled course creation and lesson publishing
- Student enrollment with wallet deduction
- Wallet top-up via voucher codes
- Video content storage and playback (recorded)
- Live streaming sessions for courses
- Real-time chat during live sessions (via Redis)
- Notification system
- Admin dashboard with control over users, courses, and reports
- PostgreSQL for core modules; Redis for chat
- Course and instructor search with filters
- Reporting on enrollments, payments, and usage
- Module-level database separation for isolation (e.g., payments)
- Event-driven communication between modules
- Live session lifecycle management
- Lesson progress tracking
- Ratings and reviews

The backend adopts a **Modular Monolith** architecture with **Node.js + Express** and **Prisma ORM**. Modules interact through **Node.js EventEmitter**, enabling loosely coupled components.

=== Key Architectural Components

- **Auth Module**: Handles OTP-based registration, login, and RBAC.
- **User Module**: Profile data, progress tracking.
- **Course Module**: Admin and instructor-based course/lesson management.
- **Enrollment Module**: Handles student enrollment, ensuring wallet deductions.
- **Payment Module**: Voucher-based top-up, wallet tracking.

- **Content Module**: Video storage metadata and secure streaming access.
- **Streaming Module**: Live session scheduling, starting, ending.
- **Chat Module**: WebSocket and Redis-based live messaging.
- **Notification Module**: Reacts to events to deliver in-app/email notifications.
- **Search Module**: Full-text PostgreSQL search.
- **Reporting Module**: Admin usage and financial metrics.
- **Admin Module**: Instructor creation, voucher management, course control.

=== Platform Rules

- Only **admins and instructors** can create or publish courses and lessons.
- **Students** can only view published content they are enrolled in.

=== Communication

All modules publish/subscribe to events via a centralized EventEmitter (e.g., `UserEnrolled`, `LessonAdded`).

== Implementation

=== Phase 1: Setup

- Set up monorepo folder structure
- Implement core dependencies: Express, Prisma, PostgreSQL, Redis, EventEmitter

=== Phase 2: Auth & User

- OTP flow
- JWT issuance
- Role verification middleware

=== Phase 3: Admin & Course

- Admin APIs to create instructors, manage courses and lessons
- Ensure only instructor/admin roles can publish

=== Phase 4: Enrollment & Payment

- Enrollment triggers wallet deduction
- Voucher top-up and transaction logging

=== Phase 5: Content & Streaming

- Upload metadata, serve signed video URLs
- Live session scheduling and streaming endpoints

=== Phase 6: Redis-Based Chat & Notifications

- Use Redis Lists for message history
- Socket.IO for WebSocket handling
- Notifications on key events

=== Phase 7: Search & Reporting

- Full-text PostgreSQL search (GIN index)
- SQL-based reporting queries

=== Phase 8: Testing & Hardening

- Load testing chat and streaming
- JWT/RBAC security checks
- Input validation

=== Phase 9: Deployment

- Docker + CI/CD
- Production PostgreSQL and Redis setup
- Prometheus/Grafana monitoring

== Milestones

=== Milestone 1: Project Bootstrapping

- Monorepo structure
- Core DB setup (SQLite dev, PostgreSQL prod)

=== Milestone 2: Auth & RBAC

- OTP login, JWT
- Role enforcement

=== Milestone 3: Admin Panel

- Instructor creation
- Course and lesson control (admin-only)

=== Milestone 4: Payments & Enrollment

- Wallet deduction
- Voucher redemption

=== Milestone 5: Content & Streaming

- S3 signed URL handling
- Live session lifecycle APIs

=== Milestone 6: Chat with Redis

- Pub/Sub chat system
- WebSocket API and Redis history

=== Milestone 7: Notifications & Reports

- Event-triggered notifications
- SQL reporting endpoints

=== Milestone 8: Search & Security

- Course/instructor search
- Rate limiting and validation

=== Milestone 9: Production Deployment

- Docker deployment
- Monitoring and CI/CD finalized

== Gathering Results

After launch, platform success will be evaluated by:

- Functional checklists (e.g., auth, enrollment, chat)
- Load testing metrics (chat, live sessions)
- Database query performance (PostgreSQL, Redis)
- Admin report accuracy and real-time analytics
- Security audit results (JWT, RBAC, OTP)

Metrics will determine MVP readiness and define next-step improvements, such as persistent chat logs, analytics dashboards, or mobile app support.