Detailed System Evaluation & Scalability Report

Date: November 28, 2025 **System:** EduLearn LMS **Version:** 2.0 (PostgreSQL Optimized)

---

1. Executive Summary

The EduLearn platform is currently at a critical juncture. While the codebase demonstrates clean coding practices and a modern tech stack (Next.js, Node.js, PostgreSQL), the system architecture contains fundamental scalability flaws that will prevent it from handling high traffic or functioning reliably on low-bandwidth networks.

Scalability Rating: ⚠ **Critical Risk** **Low-Bandwidth Performance:** ❌ **Poor**

Key Findings

1. **Database Bottlenecks:** Missing indexes on critical foreign keys will cause exponential performance degradation as data grows.
2. **API Inefficiencies:** Severe "N+1" query patterns and massive data over-fetching are present in core endpoints.
3. **Frontend Latency:** The application fetches entire course structures (including all text content) upfront, causing slow initial loads.
4. **Infrastructure Limit:** Local file storage prevents the application from scaling horizontally (adding more servers).

---

2. Detailed Technical Findings

2.1 Database Schema & Performance

Status: ⚠ **At Risk**

2.1.1 Missing Indexes

The migration script `edulearn_migrations_script.sql` creates tables but misses indexes on foreign keys used in frequent joins. This forces the database to scan entire tables to find related records.

- **Critical Missing Indexes:**
  - `course_modules(course_id)` - Required for `getCourseModules`
  - `course_lessons(module_id)` - Required for `getCourseModules`
  - `assignments(course_id)` - Required for `getStudentCourses`
  - `submissions(student_id)` - Required for `getCourseAssignments`
  - `enrollments(course_id)` - Required for `getStudentCourses`

2.1.2 Inefficient Join Patterns

File: `backend/src/controllers/courseController.ts` **Issue:** The `getCourseFiles` method uses a string manipulation join that cannot use indexes.

```
-- CURRENT IMPLEMENTATION (Extremely Slow)
JOIN course_lessons cl ON substring(cl.video_url from '[^/]+$') =
f.filename
```

Impact: This query will time out once the `course_lessons` table grows beyond a few thousand rows.

2.2 Backend API Architecture

Status: ❌ **Critical Issues**

2.2.1 N+1 Query Problem

File: `backend/src/controllers/courseController.ts` **Endpoint:** `GET /api/courses/my-courses` (`getStudentCourses`) **Issue:** The query includes a correlated subquery that executes for *every single course* a student is enrolled in.

```
(SELECT title FROM assignments a WHERE a.course_id = c.id ... LIMIT 1) as
next_deadline
```

Impact: If a student has 10 courses, the database runs 11 queries. If 1000 students load their dashboard, the database runs 11,000 queries instantly.

2.2.2 Massive Over-Fetching

File: `backend/src/controllers/courseController.ts` **Endpoint:** `GET /api/courses/:id/modules` **Issue:** The endpoint returns the full `content` (text body) and `video_url` for **every lesson** in the course at once. **Impact:** A course with 50 lessons could result in a 5MB+ JSON payload. This is devastating for mobile users on 3G networks.

2.3 Frontend Architecture

Status: ⚠ **Needs Optimization**

2.3.1 "All-or-Nothing" Data Loading

File: `app/courses/[id]/page.tsx` **Issue:** The `fetchCourseContent` function waits for Modules, Assignments, Files, and Announcements to ALL load before showing anything.

```
await Promise.all([
    authService.request(...modules),
    authService.request(...assignments),
    ...
]);
```

Impact: The user sees a loading spinner for the duration of the *slowest* request.

2.3.2 Lack of Caching

File: `lib/hooks/use-dashboard-data.ts` **Status:** ✅ **Good Start** The dashboard uses `useQuery` (React Query) with a 5-minute stale time. This is excellent. **Gap:** The Course Details page (`app/courses/[id]/page.tsx`) does **NOT** use React Query. It uses `useState` and fetches on every mount.

2.4 Infrastructure & Reliability

Status: ❌ **Not Scalable**

2.4.1 Stateful File Storage

File: `backend/src/routes/courses.ts` **Issue:** Files are saved to the local disk:

```
destination: (req, file, cb) => { path.join(__dirname, '../../uploads') }
```

Impact: You cannot run multiple backend servers behind a load balancer because a file uploaded to Server A will not be visible on Server B.

---

3. Low-Bandwidth Impact Analysis

For a user on a 2G/3G connection (e.g., 500kbps):

1. **Dashboard Load:** ~4-8 seconds (due to N+1 queries slowing down the DB response).
2. **Course Page Load:** ~15-30 seconds (due to downloading the entire course content in one massive JSON payload).
3. **Navigation:** Every time they click "Back" to dashboard and "Forward" to course, they wait the full 15-30 seconds again because there is no client-side cache for the course page.

---

4. Remediation Roadmap

Phase 1: Immediate Fixes (Week 1)

- **Database:** Execute SQL script to add missing indexes on `course_id`, `module_id`, and `student_id` columns.
- **Backend:** Rewrite `getStudentCourses` to use a `JOIN` instead of a subquery.
- **Frontend:** Refactor `CourseDetailsPage` to use `useQuery` (React Query) for caching and automatic retries.

Phase 2: Performance Tuning (Week 2)

- **Backend:** Implement "Lazy Loading" for course modules. The initial call should only return titles and IDs. Content should be fetched only when a user clicks a lesson.
- **Database:** Refactor the file association logic to avoid the `substring` join. Add a direct `file_id` to `course_lessons`.

Phase 3: Scalability (Month 1)

- **Infrastructure:** Migrate file uploads to AWS S3 or Cloudflare R2.
- **Caching:** Implement Redis to cache the "Course Structure" response, as it rarely changes but is read frequently.